IEC 61158-5-11

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 5-11: Application layer service definition – Type 11 elements

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub

The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61158-5-11

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 5-11: Application layer service definition – Type 11 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XC

ICS 35.100.70; 25.040.40

ISBN 2-8318-9458-1

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FILELDBUS SPECIFICATIONS –

Part 5-11: Application layer service definition – Type 11 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 61158-5-11 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

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 addition. This part and its Type 11 companion parts also cancel and replace IEC/PAS 62406, published in 2005.

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.

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 5-11: Application layer service definition – Type 11 elements

1 Scope

1.1 Overview

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 11 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 defines in an abstract way the externally visible service provided by the different Types of 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 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 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 do 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 fulfil any given Type of application layer services as defined in this part of IEC 61158.

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 60559, *Binary floating-point arithmetic for microprocessor systems*

IEC 61131-1, *Programmable controllers – Part 1: General information*

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

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 61158-3-11, *Industrial communication networks – Fieldbus specifications - Part 3-11: Data-link layer service definition – Type 11 elements*

ISO/IEC 646, *Information technology – ISO 7–bit coded character set for information interchange*

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model – Part 3: Naming and addressing*

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

ISO/IEC 8824, *Information Technology – Abstract Syntax notation One (ASN-1): Specification of basic notation*

ISO/IEC 8859-1, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

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

ISO/IEC 10646-1, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Architecture and Basic Multilingual Plane*

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

3 Terms and definitions, abbreviations, and conventions

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

3.1 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

3.2 ISO/IEC 8822 terms

For the purposes of this document, the following terms as defined in ISO/IEC 8822 apply:

a) abstract syntax
b) presentation context

3.3 ISO/IEC 9545 terms

For the purposes of this document, the following terms as defined in ISO/IEC 9545 apply:

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

3.4 ISO/IEC 8824 terms

For the purposes of this document, the following terms as defined in ISO/IEC 8824 apply:

a) object identifier
b) type

3.5 Fieldbus data-link layer terms

For the purposes of this document, the following terms as defined in IEC 61158-3-11 apply.

a) DLCEP
b) DLC
c) DLPDU
d) DLSDU

e) DLSAP

f) link

g) network address

h) node

i) scheduled

j) unscheduled

3.6 Fieldbus application layer type-specific definitions

For the purposes of this part of IEC 61158, the following terms and definitions apply.

3.6.1
access protection
limitation of the usage of an application object to one client

3.6.2
active connection control object
instance of a certain FAL class that abstracts the interconnection facility (as Consumer and Provider) of an automation device

3.6.3
address assignment table
mapping of the client's internal I/O-Data object storage to the decentralized input and output data objects

3.6.4
allocate
take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.6.5
application
function or data structure for which data is consumed or produced

3.6.6
application layer interoperability
capability of application entities to perform coordinated and cooperative operations using the services of the FAL

3.6.7
application objects
multiple object classes that manage and provide a run time exchange of messages across the network and within the network device

3.6.8
application process
part of a distributed application on a network, which is located on one device and unambiguously addressed

3.6.9
application process identifier
distinguishes multiple application processes used in a device

3.6.10
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 (see the definition for Application Process Object Class Definition). Application process object definitions may be accessed remotely using the services of the FAL Object Management ASE. FAL Object Management services can be used to load or update object definitions, to read object definitions, and to dynamically create and delete application objects and their corresponding definitions.

3.6.11
application process object class
a class of application process objects defined in terms of the set of their network-accessible attributes and services

3.6.12
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation. This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities

3.6.13
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.6.14
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.

3.6.15
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.

3.6.16
behaviour
indication of how an object responds to particular events

3.6.17
bit-no
designates the number of a bit in a bitstring or an octet

3.6.18
channel
single physical or logical link of an input or output application object of a server to the process

3.6.19
channel related diagnosis
information concerning a specific element of an input or output application object, provided for maintenance purposes

EXAMPLE: validity of data

3.6.20
class
a set of objects, all of which represent the same kind of system component

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

3.6.21
class attributes
attribute that is shared by all objects within the same class

3.6.22
class code
unique identifier assigned to each object class

3.6.23
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.

3.6.24
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

3.6.25
common memory
virtual common memory over the network for the Type 11 fieldbus, which is shared with the nodes participating in the Type 11 fieldbus and is primarily used for the real-time communications by the TCC data service

3.6.26
communication objects
components that manage and provide a run time exchange of messages across the network

EXAMPLES: Connection Manager object, Unconnected Message Manager (UCMM) object, and Message Router object

3.6.27
configuration check
comparison of the expected I/O-Data object structuring of the client with the real I/O-Data object structuring to the server in the start-up phase

3.6.28
configuration data base
interconnection information maintained by the ACCO ASE

3.6.29
configuration fault
an unacceptable difference between the expected I/O-Data object structuring and the real I/O-Data object structuring, as detected by the server

3.6.30
configuration identifier
representation of a portion of I/O Data of a single input- and/or output-module of a server

3.6.31
connection
logical binding between application objects that may be within the same or different devices

NOTE 1 Connections may be either point-to-point or multipoint.

NOTE 2 The logical link between sink and source of attributes and services at different custom interfaces of RT-Auto ASEs is referred to as interconnection. There is a distinction between data and event interconnections. The logical link and the data flow between sink and source of automation data items is referred to as data interconnection. The logical link and the data flow between sink (method) and source (event) of operational services is referred to as event interconnection.

3.6.32
connection channel
description of a connection between a sink and a source of data items

3.6.33
connection ID (CID)
identifier assigned to a transmission that is associated with a particular connection between producers and consumers, providing a name for a specific piece of application information

3.6.34
connection path
an octet stream that defines the application object to which a connection instance applies

3.6.35
connection point
buffer which is represented as a subinstance of an Assembly object

3.6.36
consume
act of receiving data from a producer

3.6.37
consumer
node or sink that is receiving data from a producer

3.6.38
consuming application
application that consumes data

3.6.39
consumerID
unambiguous identifier within the scope of the ACCO assigned by the consumer to recognize the internal data of a configured interconnection sink

3.6.40
control commands
action invocations transferred from client to server to clear outputs, freeze inputs and/or synchronize outputs

3.6.41
conveyance path
unidirectional flow of APDUs across an application relationship

3.6.42
cyclic
repetitive in a regular manner

3.6.43
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.6.44
data marshalling
the encoding of parameters of the FAL service primitives with respect to their interface definition

<Type 10> NOTE This is part of the abstract ORPC model.

3.6.45
dedicated AR
AR used directly by the FAL User

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

3.6.46
default DL-address
value 126 as an initial value for DL-address, which has to be changed (e.g. by assignment of an DL-address via the fieldbus) before operation with a DP-master (class 1)

3.6.47
device
physical hardware connected to the link

NOTE A device may contain more than one node.

3.6.48
device profile
a collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.6.49
diagnosis information
all data available at the server for maintenance purposes

3.6.50
diagnosis information collection
system diagnosis information that is assembled at the client side

3.6.51
dynamic AR
AR that requires the use of the AR establishment procedures to place it into an established state

3.6.52
end node
producing or consuming node

3.6.53
endpoint
one of the communicating entities involved in a connection

3.6.54
engineering
abstract term that characterizes the client application or device responsible for configuring an automation system via interconnecting data items

3.6.55
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.6.56
error class
general grouping for related error definitions and corresponding error codes

3.6.57
error code
identification of a specific type of error within an error class

3.6.58
event
an instance of a change of conditions

3.6.59
FAL subnet
subnetworks composed of one or more data link segments, identified by a subset of the network address

NOTE FAL subnets are permitted to contain bridges but not routers.

3.6.60
FIFO variable
a Variable Object class, composed of a set of homogeneously typed elements, where the first written element is the first element that can be read

NOTE On the fieldbus only one, complete element can be transferred as a result of one service invocation.

3.6.61
frame
denigrated synonym for DLPDU

3.6.62
group
a) <general> a general term for a collection of objects. Specific uses:

b) <addressing> when describing an address, an address that identifies more than one entity

3.6.63
interface
(a) shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

(b) collection of FAL class attributes and services that represents a specific view on the FAL class

3.6.64
interface definition language
syntax and semantics of describing service parameters in a formal way

NOTE This description is the input for the ORPC model, especially for the ORPC wire protocol.

3.6.65
interface pointer
key attribute that unambiguously addresses an object interface instance

3.6.66
invocation
act of using a service or other resource of an application process

NOTE Each invocation represents a separate thread of control that may be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a service that has been initiated but not yet completed is referred to as an outstanding service invocation.

3.6.67
I/O data
object designated to be transferred cyclically for the purpose of processing

3.6.68
identifier related diagnosis
information dedicated to modules for maintenance purpose

3.6.69
index
address of an object within an application process

3.6.70
instance
the actual physical occurrence of an object within a class that identifies one of many objects within the same object class

EXAMPLE California is an instance of the object class state.

NOTE The terms object, instance, and object instance are used to refer to a specific instance.

3.6.71
instance attributes
attribute that is unique to an object instance and not shared by the object class

3.6.72
instantiated
object that has been created in a device

3.6.73
logical device
a certain FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.6.74
manufacturer ID
identification of each product manufacturer by a unique number

3.6.75
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.

3.6.76
master parameter set
the configuration and parameterization data of all DP-slaves that are assigned to the corresponding DP-master and the bus parameters

3.6.77
member
piece of an attribute that is structured as an element of an array

3.6.78
message router
object within a node that distributes messaging requests to appropriate application objects

3.6.79
method
<object> a synonym for an operational service which is provided by the server ASE and invoked by a client

3.6.80
module
a) <general> hardware or logical component of a physical device

b) <Type 3> addressable unit inside the DP-slave

3.6.81
multipoint connection
connection from one node to many

NOTE Multipoint connections allow messages from a single producer to be received by many consumer nodes.

3.6.82
network
a set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.6.83
object
abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behaviour

3.6.84
object remote procedure call
model for object oriented or component based remote method invocation

3.6.85
object specific service
service unique to the object class which defines it

3.6.86
originator
client responsible for establishing a connection path to the target

3.6.87
peer
role of an AR endpoint in which it is capable of acting as both client and server

3.6.88
physical device
<general> an automation or other network device

<Type10> a certain FAL class that abstracts the hardware facilities of an automation device

3.6.89
point-to-point connection
connection that exists between exactly two application objects

3.6.90
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

3.6.91
pre-established AR endpoint
AR endpoint that is placed in an established state during configuration of the AEs that control its endpoints

3.6.92
process data
object(s) which are already pre-processed and transferred acyclically for the purpose of information or further processing

3.6.93
produce
act of sending data to be received by a consumer

3.6.94
producer
node that is responsible for sending data

3.6.95
property
a general term for descriptive information about an object

3.6.96
provider
source of a data connection

3.6.97
providerID
an unambiguous identifier within the scope of the ACCO assigned by the provider to recognize the internal data of a configured interconnection source

3.6.98
publisher
role of an AR endpoint that transmits APDUs onto the fieldbus for consumption by one or more subscribers

NOTE A publisher may not be aware of the identity or the number of subscribers and it may publish its APDUs using a dedicated AR.

3.6.99
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

3.6.100
pull publisher
type of publisher that publishes an object in response to a request received from its pull publishing manager

3.6.101
pull publishing manager
type of publishing manager that requests that a specified object be published in a corresponding response APDU

3.6.102
push publisher
type of publisher that publishes an object in an unconfirmed service request APDU

3.6.103
push publishing manager
type of publishing manager that requests that a specified object be published using an unconfirmed service

3.6.104
pull subscriber
type of subscriber that recognizes received confirmed service response APDUs as published object data

3.6.105
push subscriber
type of subscriber that recognizes received unconfirmed service request APDUs as published object data

3.6.106
quality code aware
attribute of the RT-Auto class that indicates that an RT-Auto object uses a status code for its data items

3.6.107
quality code
additional status information of a data item

3.6.108
quality code unaware
opposite of quality code aware

3.6.109
real configuration
input and output data structure of the DP-slave, including definition of data consistency

3.6.110
resource
a processing or information capability of a subsystem

3.6.111
route endpoint
object container containing Variable Objects of a variable class

3.6.112
RT-auto
an FAL class that abstracts the automation function as a process-related component of an automation device

3.6.113
runtime object model
objects that exist in a device together with their interfaces and methods that are accessible

3.6.114
serial number
<Type 2> a unique 32-bit integer value assigned by each manufacturer to every device having Type 2 communication capabilities

NOTE The Manufacturer ID and serial number jointly form a unique identifier for each device.

3.6.115
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

3.6.116
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.6.117
slot
address of a module within a DP-slave

3.6.118
subscriber
role of an AREP in which it receives APDUs produced by a publisher

3.6.119
sync
function at the DP-slaves for simultaneous data transfer between the output data object and the process

3.6.120
target
end-node to which a connection is established

3.6.121
unconnected message manager (UCMM)
component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object

3.6.122
unconnected service
messaging service which does not rely on the set up of a connection between devices before allowing information exchanges

3.7 Abbreviations and symbols

ACCO	Active connection control object
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 endpoint
ASCII	American Standard Code for Information Interchange
ASE	Application service element

CID	Connection ID
CIM	Computer integrated manufacturing
CIP	Control and information protocol
CM	Common memory
CM_API	Actual packet interval
CM_RPI	Requested packet interval
Cnf	Confirmation
COR	Connection originator
CR	Communication relationship
CREP	Communication relationship endpoint
DL-	(as a prefix) data-link-
DLC	Data-link connection
DLCEP	Data-link connection endpoint
DLL	Data-link layer
DLM	Data-link-management
DLSAP	Data-link service access point
DLSDU	DL-service-data-unit
DNS	Domain name service
DP	Decentralised peripherals
FAL	Fieldbus application layer
FIFO	First-in First-out
HMI	Human-machine interface
ID	Identifier
IDL	Interface definition language
IEC	International Electrotechnical Commission
Ind	Indication
IP	Internet protocol
ISO	International Organization for Standardization
LDev	Logical device
LME	Layer management entity
O2T	Originator to target (connection characteristics)
O⇒T	Originator to target (connection characteristics)
ORPC	Object remote procedure call
OSI	Open Systems Interconnect
PDev	Physical device
PDU	Protocol data unit
PL	Physical layer
QoS	Quality of service
QC	Quality code
REP	Route endpoint
Req	Request
Rsp	Response

RT	Runtime
SAP	Service access point
SCL	Security level
SDU	Service data unit
SEM	State event matrix
SMIB	System management information base
SMK	System management kernel
STD	State transition diagram, used to describe object behaviour
S-VFD	Simple virtual field device
T2O	Target to originator (connection characteristics)
T⇒O	Target to originator (connection characteristics)
VAO	Variable object

3.8 Conventions

3.8.1 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.

3.8.2 General conventions

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

3.8.3 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 is used to specify

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 is used to specify services conditional on a constraint statement.

3.8.4 Conventions for service definitions

3.8.4.1 General

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

3.8.4.2 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 is used to specify 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.

3.8.4.3 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.

3.8.5 Type 11 specific conventions

None.

3.9 Nomenclature for references within this standard

Clauses, including annexes, can be referenced in their entirety, including any subordinate subclauses, as “clause N” or “Annex N”, where N is the number of the clause or letter of the annex.

Subclauses can be referenced in their entirety, including any subordinate subclauses, as “N.M” or “N.M.P” and so forth, depending on the level of the subclause, where N is the number of the subclause or letter of the annex, and M, P and so forth represent the successive levels of subclause up to and including the subclause of interest.

When a clause or subclause contains one or more subordinate subclauses, the text between the clause or subclause heading and its first subordinate subclause can be referenced in its entirety as “N.0” or “N.M.0” or “N.M.P.0” and so forth, where N, M and P are as above. Stated differently, a reference ending with “.0” designates the text and figures between a clause or subclause header and its first subordinate subclause.

4 Concepts

4.1 Common concepts

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

4.2 Type specific concepts

4.2.1 General

This standard specifies the Application layer service of Type 11 essential for the ISO/IEC 8802-3-based Time-critical Control Network (TCnet), which is one of the communication networks for the Real- Time Ethernet (RTE) defined in the IEC 61784-2 and is referred to as RTE-TCnet hereafter.

The Type 11 fieldbus meets the industrial automation market objective of providing predictable time deterministic and reliable time-critical data transfer and means, which allow co-existence with non-time-critical data transfer over the ISO/IEC 8802-3 series communications medium, for support of cooperation and synchronization between automation processes on field devices in a real-time application system. 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.

This standard specifies the part of the protocol set of the RTE-TCnet communication profile and/or of one or more communication profiles related to a common family of the RTE-TCnet. The RTE-TCnet communication profile, shown in Figure 1 as one of the profile sets, is based on the 7 layer OSI Basic Reference model. For the regular ISO/IEC 8802-3-based applications, the upper layers mapped over the data-link layer is in the ordinary way; on the other hand, for the time-critical applications with the common-memory running in parallel, the specific application layer for the RTE-TCnet is specified. The data-link layer for the RTE-

TCnet has the extension, but is compliant to the ISO/IEC 8802-3 MAC protocol in order to provide both services for the time-critical communications and the common-memory applications respectively.

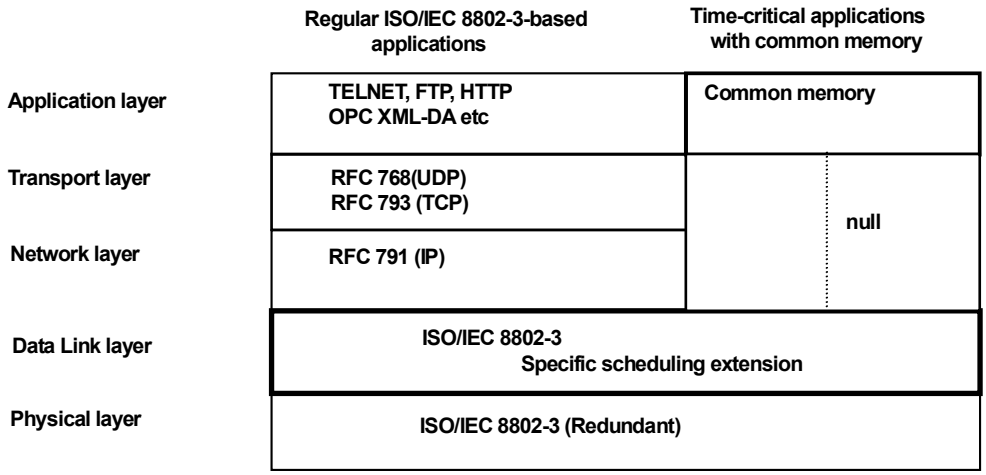

Figure 1 – RTE-TCnet communication profile

This standard specifies the data-link protocol as the essential parts of the RTE-TCnet profile, which are the extension part of the ISO/IEC 8802-3-based data-link layer and the Application layer exploiting the services of the data-link layer immediately below, in terms of the "three-layer" Fieldbus Reference Model which is based in part on the OSI Basic Reference Model. Other part of RTE-TCnet profile is not in the scope of this document.

4.2.1.1 Field of applications

In industrial control systems, several kinds of field devices such as drives, sensors and actuators, programmable-controllers, distributed-control systems and human-machine interface devices are required to be connected with control networks. The process control data and the state data is transferred among these field devices in the system and the communications between these field devices requires simplicity in application programming and to be executed with adequate response time. In most industrial automation systems such as food, water, sewage, paper and steel, including a rolling mill, the control network is required to provide time-critical response capability for their application, as required in the ISO/TR 13283 for time-critical communications architectures.

Plant production may be compromised due to errors, which could be introduced to the control system if the network does not provide a time-critical response. Therefore the following characteristics are required for a time-critical control network:

– a deterministic response time between the control device nodes;

– ability to share process data seamlessly across the control system.

The RTE-TCnet is applicable to such industrial automation environment, in which time-critical communications is primarily required. 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.

4.2.2 Overview of the Common memory (CM)

The RTE-TCnet Application layer service utilizes the RTE-TCnet specific common-memory system, so-called Common Memory (CM). The CM is a virtual common-memory over the RTE-TCnet, is used and globally shared by the participating node, the application processes

running over each node. Further the CM by means of the time-critical cyclic data transfer services which is specified in the IEC 61158-3-11 Clause 4, provides the data distribution in temporal and spatial coherency.

The size and capacity depends on the implementation. However the CM is divided into numbers of block with several size of memory. The number and the size depends also on implementation. The time-critical cyclic data transfer, specified by the DL-service and the DL-protocol in this specification, is carried out on each data block basis and each data of block is multicasted to the member nodes from a node as publisher.

Each block in the CM is associated with one of Application Relationship End Point (AREP), and is identified by the AREP and is used by multiple application processes in common. The block is a container for application data in general use and provides flexibility to apply in a variety of industrial application processes.

Figure 2 shows one of the application examples by using the CM.

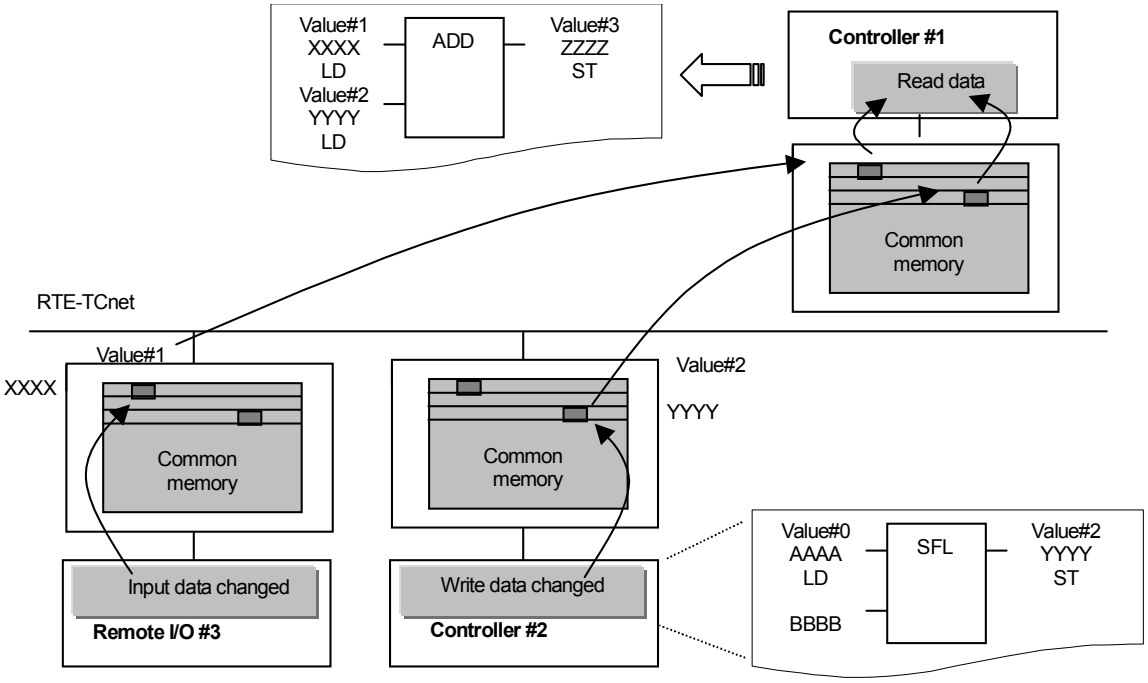

Figure 2 – Application example by using the CM

4.2.3 Common memory (CM) concept

Figure 3 shows the concept of the RTE-TCnet cyclic data transmission with the CM. It utilizes a cyclic broadcast transmission mechanism with the CM that is actually implemented in each node and given the same address space on the network. The CM is divided into dedicated areas for each node's transmitting data. That is refreshed in the same memory area of all nodes on a fixed cyclic period. By this means, the controllers can quickly access each other's data avoiding troublesome communication procedures.

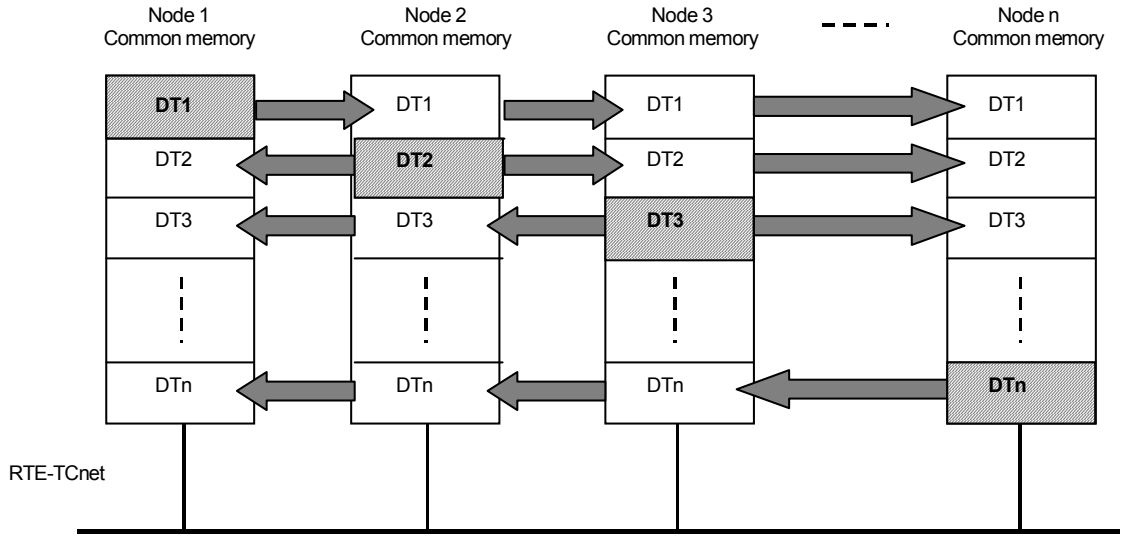

Figure 3 – Global common-memory concept over the RTE-TCnet

4.2.4 Relationship of common memory and AREP

The CM is divided into numbers of block with several size of memory, of which number and size depends on the implementation; however, the size is recommended from 16 to 64 for efficient application.

Each block in the CM is associated with one Application Relationship End Point (AREP), which is unique, is commonly used and identified in the RTE-TCnet domain. The unique number assigned to each block associated with one of AREP is used to identify and determine actual position of the CM address.

Each node is assigned a number of blocks of AREPs as Publisher, and broadcasts data each block, receives data from each block from other node as a subscriber and updates the contents of the corresponding block on the local physical memory which is identical configuration to the CM.

When on creation of new AREP, AL-user specifies three kind of class, that is high-speed、medium-speed and low-speed class, to the AREP.

Figure 4 depicts the relationship between the CM and the AREP on each node.

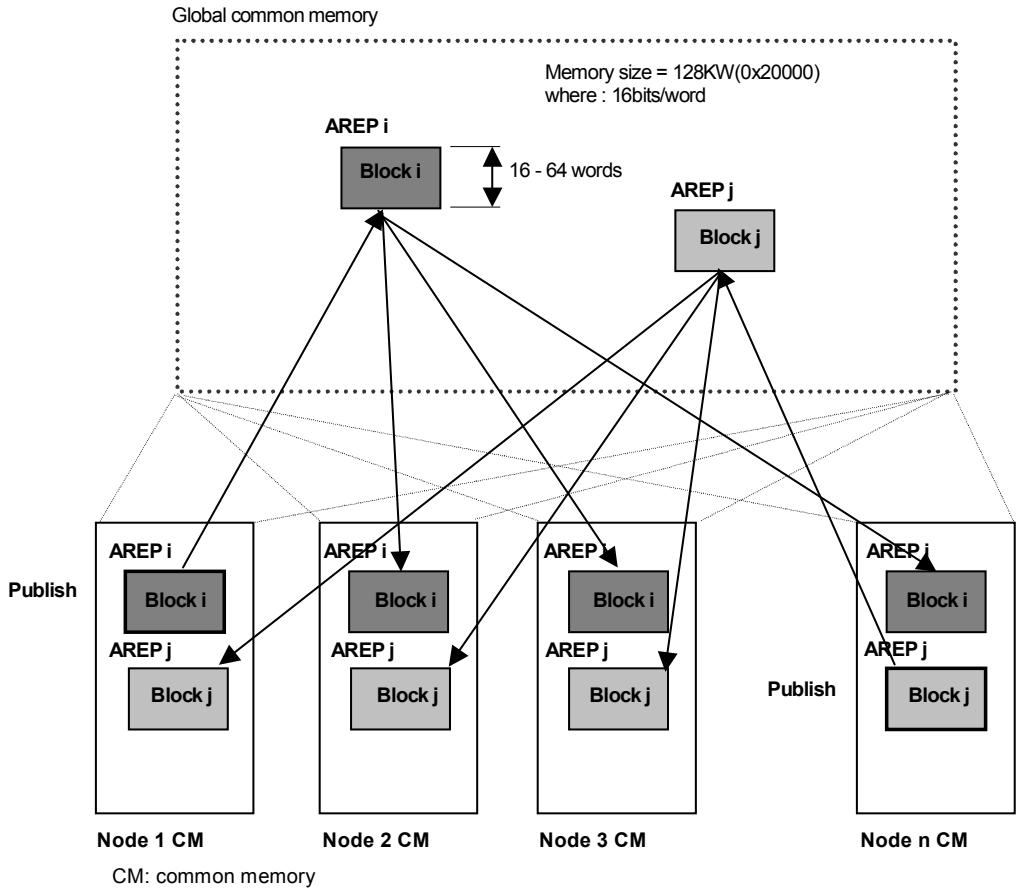

Figure 4 – Relationship of Common Memory and AREP

4.2.5 Common memory data type

The data type applied to a block which is associated with one of AREP, is primarily based on the basic data type defined in IEC 61158-5 Clause 5. Using these data type, Array or Structure is built.

The primitive data types forming Array or Structure in a block of the CM is as follows:

Boolean;
BitString8;
BitString16;
BitSting32;
BinaryDate2000;
DATE;
TimeOfDay without date indication;
TimeDifference without date indication;
Integer8;
Integer16;
Integer32;
Unsigned8;
Unsigned16;
Unsigned32;

Float32;
BinaryTime0;
BinaryTime1;
BinaryTime2;
BinaryTime3;
BinaryTime4;
BinaryTime5;
BinaryTime6;
BinaryTime7;
OctetString2;
OctetString4;
VisibleString2;
VisibleString4.

4.2.6 Type 11 ASE and services

The Type 11 Application layer provides the Update_memory service for updating the contents of the CM. For this purpose, the Type 11 AE is provided with the CM ASE and the AR ASE.

Figure 5 depicts the structure of the Type 11 AL ASE.

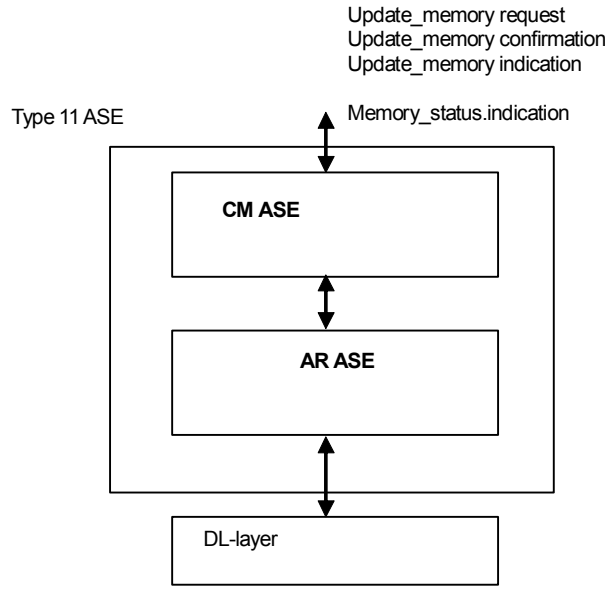

Figure 5 – Structure of Type 11 AL ASE

5 Data type ASE

5.1 General

5.1.1 Overview

Fieldbus data types specify the machine independent syntax for application data conveyed by FAL services. The fieldbus application layer supports the definition and transfer of both basic and constructed data types. Encoding rules for the data types specified in this Clause are provided in IEC 61158-6-11.

Basic types are atomic types that cannot be decomposed into more elemental types. Constructed types are types composed of basic types and other constructed types. Their complexity and depth of nesting is not constrained by this standard.

Data types are defined in IEC 61158-6-11 as instances of the “Data Type” class shown in Figure 6. Only a subset of the IEC 61158 data types are shown in this figure. Defining new types is accomplished by providing a numeric id and supplying values for the attributes defined for the data type class.

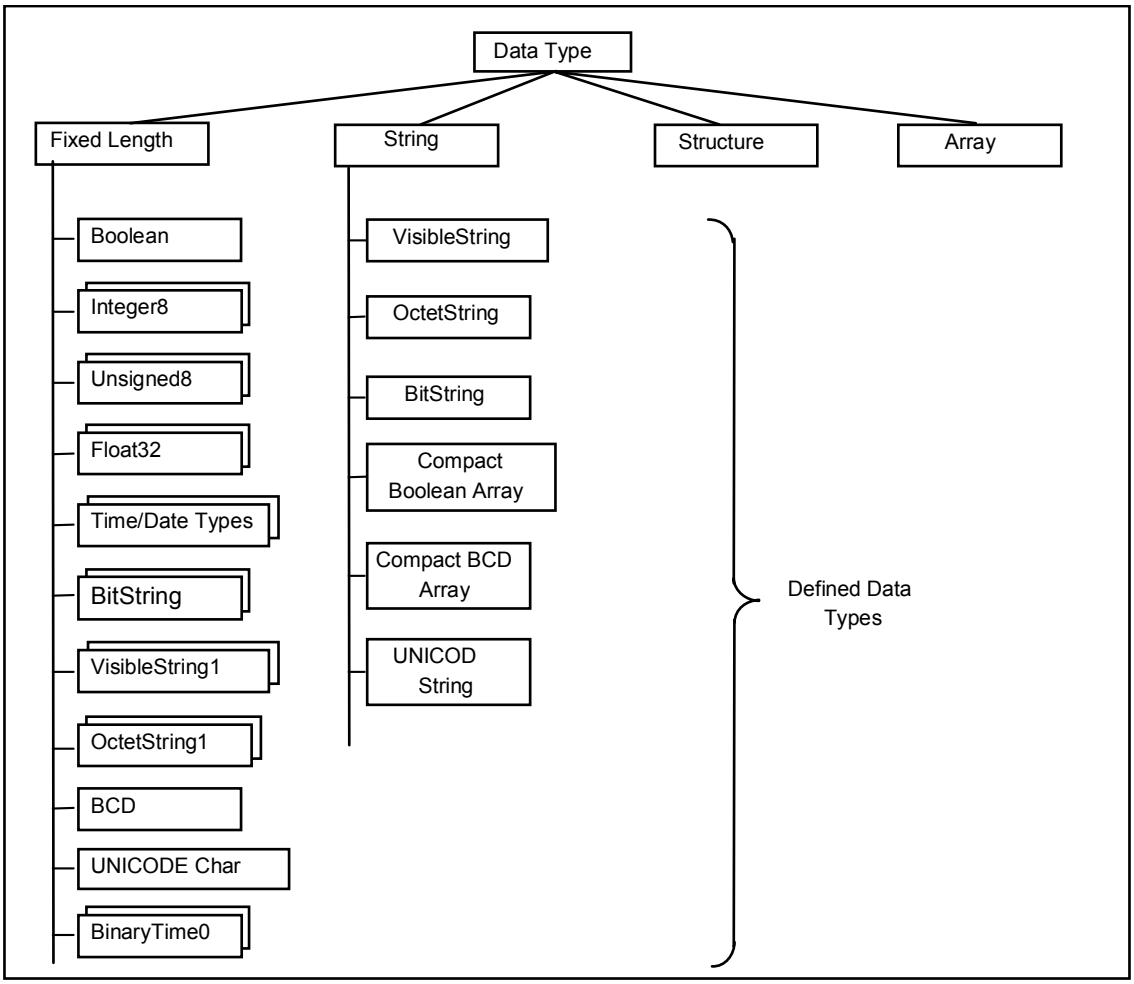

Figure 6 – Data type class hierarchy example

The basic data classes are used to define fixed length and bitstring data types. Standard types taken from ISO/IEC 8824 are referred to as *simple* data types. Other standard basic data types are defined specifically for fieldbus applications and are referred to as *specific types*.

The constructed types specified in this standard are strings, arrays and structures. There are no standard types defined for arrays and structures.

5.1.2 Basic type overview

Most basic types are defined from a set of ISO/IEC 8824 types (simple types). Some ISO/IEC 8824 types have been extended for fieldbus specific use (specific types).

Simple types are ISO/IEC 8824 universal types. They are defined in this standard to provide them with fieldbus class identifiers.

Specific types are basic types defined specifically for use in the fieldbus environment. They are defined as simple class subtypes.

Basic types have a constant length. Two variations are defined, one for defining data types whose length is an integral number of octets, and one for defining data types whose length is bits.

NOTE Boolean, Integer, OctetString, VisibleString, and UniversalTime are defined in this standard for the purpose of assigning fieldbus class identifiers to them. This standard does not change their definitions as specified in ISO/IEC 8824.

5.1.3 Fixed length type overview

The length of Fixed length types is an integral number of octets.

5.1.4 Constructed type overview

5.1.4.1 Strings

A string is composed of an ordered set, variable in number, of homogeneously typed fixed-length elements.

5.1.4.2 Arrays

An array is composed of an ordered set of homogeneously typed elements. This standard places no restriction on the data type of array elements, but it does require that each element be of the same type. Once defined, the number of elements in an array may not be changed.

5.1.4.3 Structures

A structure is made of an ordered set of heterogeneously typed elements called fields. Like arrays, this standard does not restrict the data type of fields. However, the fields within a structure do not have to be of the same type.

5.1.4.4 Nesting level

This standard permits arrays and structures to contain arrays and structures. It places no restriction on the number of nesting levels allowed. However, the FAL services defined to access data provide for partial access to the level negotiated by the Initiate service. The default number of levels for partial access is one.

When an array or structure contains constructed elements, access to a single element in its entirety is always provided. Access to subelements of the constructed element is also provided, but only when explicitly negotiated during AR establishment, or when explicitly preconfigured on pre-established ARs.

NOTE For example, suppose that a data type named "employee" is defined to contain the structure "employee name", and "employee name" is defined to contain "last name" and "first name". To access the "employee" structure, the FAL permits independent access to the entire structure and to the first level field "employee name". Without explicitly negotiating partial access to more than one level, independent access to "last name" or "first name" would not be possible; their values could only be accessed together as a unit through access to "employee" or "employee name".

5.1.5 Specification of user defined data types

Users may find it necessary to define custom data types for their own applications. User defined types are supported by this standard as instances of data type classes.

User defined types are specified in the same manner that all FAL objects are specified. They are defined by providing values for the attributes specified for their class.

5.1.6 Transfer of user data

User data is transferred between applications by the FAL protocol. All encoding and decoding are performed by the FAL user.

The rules for encoding user data in FAL protocol data units is data type dependent. These rules are defined in IEC 61158-6-11. User-defined data types for which there are no encoding rules are transferred as a variable-length sequence of octets. The format of the data within the octet string is defined by the user.

5.2 Formal definition of data type objects

5.2.1 Data type class

5.2.1.1 Template

The data type class specifies the root of the data type class tree. Its parent class "top" indicates the top of the FAL class tree.

FAL ASE:			**DATA TYPE ASE**
CLASS:			**DATA TYPE**
CLASS ID:			5 (FIXED LENGTH & STRING), 6 (STRUCTURE), 12 (ARRAY)
PARENT CLASS:			TOP
ATTRIBUTES:			
1	(o)	Key Attribute:	Data Type Numeric Identifier
2	(o)	Key Attribute:	Data Type Name
3	(m)	Attribute:	Format (FIXED LENGTH, STRING, STRUCTURE, ARRAY)
4	(c)	Constraint:	Format = FIXED LENGTH | STRING
4.1	(m)	Attribute:	Octet Length
5	(c)	Constraint:	Format = STRUCTURE
5.1	(m)	Attribute:	Number of Fields
5.2	(m)	Attribute:	List of Fields
5.2.1	(o)	Attribute:	Field Name
5.2.2	(m)	Attribute:	Field Data Type
6	(c)	Constraint:	Format = ARRAY
6.1	(m)	Attribute:	Number of Array Elements
6.2	(m)	Attribute :	Array Element Data Type

5.2.1.2 Attributes

Data type numeric identifier

This attribute identifies the numeric identifier of the related data type. Each IEC 61158-5*tt* part defines its own numeric identifiers, and there is no requirement for data type numeric identifiers to be unique across the various IEC 61158-5*tt* parts.

Data type name

This optional attribute identifies the name of the related data type.

Format

This attribute identifies the data type as a fixed-length, string, array, or data structure.

Octet length

This conditional attribute defines the representation of the dimensions of the associated type object. It is present when the value of the format attribute is "FIXED LENGTH" or "STRING". For FIXED LENGTH data types, it represents the length in octets. For STRING data types, it represents the length in octets for a single element of a string.

Number of fields

This conditional attribute defines the number of fields in a structure. It is present when the value of the format attribute is "STRUCTURE".

List of fields

This conditional attribute is an ordered list of fields contained in the structure. Each field is specified by its number and its type. Fields are numbered sequentially from 0 (zero) in the order in which they occur. Partial access to fields within a structure is supported by identifying the field by number. This attribute is present when the value of the format attribute is "STRUCTURE".

Field name
This conditional, optional attribute specifies the name of the field. It may be present when the value of the format attribute is "STRUCTURE".

Field data type
This conditional attribute specifies the data type of the field. It is present when the value of the format attribute is "STRUCTURE". This attribute may itself specify a constructed data type either by referencing a constructed data type definition by its numeric id, or by embedding a constructed data type definition here. When embedding a description, the Embedded Data Type description shown below is used.

Number of array elements

This conditional attribute defines the number of elements for the array type. Array elements are indexed starting at “0” through “n-1” where the size of the array is “n” elements. This attribute is present when the value of the format attribute is "ARRAY".

Array element data type

This conditional attribute specifies the data type for the elements of an array. All elements of the array have the same data type. It is present when the value of the format attribute is "ARRAY". This attribute may itself specify a constructed data type either by referencing a constructed data type definition by its numeric id, or by embedding a constructed data type definition here. When embedding a description, the Embedded Data Type description shown below is used.

Embedded data type description

This attribute is used to recursively define embedded data types within a structure or array. The template below defines its contents. The attributes shown in the template are defined above in the data type class, except for the Embedded Data Type attribute, which is a recursive reference to this attribute. It is used to define nested elements.

ATTRIBUTES:

1	(m)	Attribute:	Format(FIXED LENGTH, STRING, STRUCTURE, ARRAY)
2	(c)	Constraint:	Format = FIXED LENGTH | STRING
2.1	(m)	Attribute:	Data Type Numeric ID value
2.2	(m)	Attribute:	Octet Length
3	(c)	Constraint:	Format = STRUCTURE
3.1	(m)	Attribute:	Number of Fields
3.2	(m)	Attribute:	List of Fields
3.2.1	(m)	Attribute:	Embedded Data Type Description
4	(c)	Constraint:	Format = ARRAY
4.1	(m)	Attribute:	Number of Array Elements
4.2	(m)	Attribute:	Embedded Data Type Description

5.3 FAL defined data types

5.3.1 Fixed length types

5.3.1.1 Boolean types

5.3.1.1.1 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 BOOL

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

5.3.1.1.3 VT_BOOLEAN

CLASS: **Data Type**

ATTRIBUTES:

2	Data Type Name	=	VT_BOOLEAN
4	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This data type expresses a Boolean data type with the values TRUE (-1) and FALSE (0) (see Interger16).

5.3.1.2 Bitstring types

5.3.1.2.1 BitString8

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	22
2	Data Type Name	=	Bitstring8
3	Format	=	FIXED LENGTH
5.1	Octet Length	=	1

This type contains 1 element of type BitString.

5.3.1.2.2 OCTET

This IEC 61131-3 type is the same as Bitstring8.

5.3.1.2.3 BitString16

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	23
2	Data Type Name	=	Bitstring16
3	Format	=	FIXED LENGTH
5.1	Octet Length	=	2

5.3.1.2.4 WORD

This IEC 61131-3 type is the same as Bitstring16.

5.3.1.2.5 BitString32

CLASS:			Data Type
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	24
2	Data Type Name	=	Bitstring32
3	Format	=	FIXED LENGTH
5.1	Octet Length	=	4

5.3.1.2.6 DWORD

This IEC 61131-3 type is the same as Bitstring32.

5.3.1.2.7 BitString64

CLASS:			Data Type
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	57
2	Data Type Name	=	Bitstring64
3	Format	=	FIXED LENGTH
5.1	Octet Length	=	8

5.3.1.2.8 LWORD

This IEC 61131-3 type is the same as Bitstring64.

5.3.1.3 Currency types

5.3.1.3.1 currency

CLASS:	Data Type		
ATTRIBUTES:			
2	Data Type Name	=	currency
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This data type defines a signed 64-bit integer in units of 1/10,000 (or 1/100 of a cent). A currency number stored as an 8-octet, two's complement integer, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. This representation provides a range of ±922337203685477,5807. This data type is useful for calculations involving money, or for any fixed-point calculation where accuracy is particularly important.

5.3.1.4 Date/Time types

5.3.1.4.1 BinaryDate

CLASS:			Data Type
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	11
2	Data Type Name	=	BinaryDate
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	7

This data type is composed of six elements of unsigned values and expresses calendar date and time. The first element is an Unsigned16 data type and gives the fraction of a minute in milliseconds. The second element is an Unsigned8 data type and gives the fraction of an hour in minutes. The third element is an Unsigned8 data type and gives the fraction of a day in hours. The fourth element is an Unsigned8 data type. Its upper three (3) bits give the day of the week and its lower five (5) bits give the day of the month. The fifth element is an Unsigned8 data type and gives the month. The last element is Unsigned8 data type and gives the year.

5.3.1.4.2 BinaryDate2000

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	51
2	Data Type Name	=	BinaryDate2000
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This data type is composed of six elements of unsigned values and expresses calendar date and time. The first element is an Unsigned16 data type and gives the fraction of a minute in milliseconds. The second element is an Unsigned8 data type and gives the fraction of an hour in minutes. The third element is an Unsigned8 data type and gives the fraction of a day in hours. The fourth element is an Unsigned8 data type. Its upper three (3) bits give the day of the week and its lower five (5) bits give the day of the month. The fifth element is an Unsigned8 data type and gives the month. The last element is Unsigned16 data type and gives the year.

5.3.1.4.3 Date

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	50
2	Data Type Name	=	Date
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	7

This data type is composed of six elements of unsigned values and expresses calendar date and time. The first element is an Unsigned16 data type and gives the fraction of a minute in milliseconds. The second element is an Unsigned8 data type and gives the fraction of an hour in minutes. The third element is an Unsigned8 data type and gives the fraction of a day in hours with the most significant bit indicating Standard Time or Daylight Saving Time. The fourth element is an Unsigned8 data type. Its upper three (3) bits give the day of the week and its lower five (5) bits give the day of the month. The fifth element is an Unsigned8 data type and gives the month. The last element is Unsigned8 data type and gives the year. The values 0 … 50 correspond to the years 2000 to 2050, the values 51 … 99 correspond to the years 1951 to 1999.

5.3.1.4.4 DATE

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	DATE
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This IEC 61131-3 type is a binary number. The most significant bit of the most significant octet 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. It expresses the date as a number of days, starting from 1972.01.01 (January 1st, 1972), the start of the Coordinated Universal Time (UTC) era, until 2151.06.06 (June 6th, 2151), i.e. a total range of 65536 days.

5.3.1.4.5 date

This data type is the same as Float64.

The data type date has a resolution in the range of one nanosecond. It is valid for dates between 1 January 0100 and 31 December 9999. The value 0,0 has been defined for 30 December 1899, 00:00. The integer part of the value represents the days after 30 December

1899 (for dates before this day, the corresponding value is negative); the fractional part defines the time at that day.

5.3.1.4.6 TimeOfDay

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	12
2	Data Type Name	=	TimeOfDay
4	Format	=	FIXED LENGTH
4.1	Octet Length	=	6

This data type is composed of two elements of unsigned values and expresses the time of day and the date. The first element is an Unsigned32 data type and gives the time after the midnight in milliseconds. The second element is an Unsigned16 data type and gives the date counting the days from January 1, 1984.

5.3.1.4.7 TimeOfDay with date indication

This data type is is the same as the TimeOfDay data type defined above.

5.3.1.4.8 TimeOfDay without date indication

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	52
2	Data Type Name	=	TimeOfDay without date indication
4	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This data type is composed of one element of an unsigned value and expresses the time of day. The element is an Unsigned32 data type and gives the time after the midnight in milliseconds.

5.3.1.4.9 TIME_OF_DAY

This IEC 61131-3 type is the same as TimeofDay without date indication.

5.3.1.4.10 TimeDifference

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	13
2	Data Type Name	=	TimeDifference
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4 or 6

This data type is composed of two elements of unsigned values that express the difference in time. The first element is an Unsigned32 data type that provides the fractional portion of one day in milliseconds. The optional second element is an Unsigned16 data type that provides the difference in days.

5.3.1.4.11 TimeDifference with date indication

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	53
2	Data Type Name	=	TimeDifference with date indication
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	6

This data type is composed of two elements of unsigned values that express the difference in time. The first element is an Unsigned32 data type that provides the fractional portion of one day in milliseconds. The second element is an Unsigned16 data type that provides the difference in days.

5.3.1.4.12 TimeDifference without date indication

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	54
2	Data Type Name	=	TimeDifference without date indication
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This data type is composed of one element of an unsigned value that express the difference in time. The element is an Unsigned32 data type that provides the fractional portion of one day in milliseconds.

5.3.1.4.13 TimeValue

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	21
2	Data Type Name	=	Time Value
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This simple type expresses the time or time difference in a two's complement binary number with a length of eight octets. The unit of time is 1/32 millisecond.

5.3.1.4.14 UniversalTime

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	16
2	Data Type Name	=	UniversalTime
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	12

This simple type is composed of twelve elements of type VisibleString. (YYMMDDHHMMSS). It is the same as that defined in ISO/IEC 8824, except that the local time differential is not supported.

5.3.1.4.15 FieldbusTime

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	17
2	Data Type Name	=	FieldbusTime
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	7

NOTE This data type is defined in IEC 61158-4 of this standard as DL-Time.

5.3.1.4.16 BinaryTime0

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	40
2	Data Type Name	=	BinaryTime0
3	Format	=	FIXED LENGTH

4.1 Octet Length = 2

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of two octets. The unit of time for this type is 10 µs.

5.3.1.4.17 BinaryTime1

CLASS: **Data Type**

ATTRIBUTES:

1 Data Type Numeric Identifier = 41

2 Data Type Name = BinaryTime1

3 Format = FIXED LENGTH

4.1 Octet Length = 2

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of two octets. The unit of time for this type is 100 µs.

5.3.1.4.18 BinaryTime2

CLASS: **Data Type**

ATTRIBUTES:

1 Data Type Numeric Identifier = 42

2 Data Type Name = BinaryTime2

3 Format = FIXED LENGTH

4.1 Octet Length = 2

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of two octets. The unit of time for this type is 1 ms.

5.3.1.4.19 BinaryTime3

CLASS: **Data Type**

ATTRIBUTES:

1 Data Type Numeric Identifier = 43

2 Data Type Name = BinaryTime3

3 Format = FIXED LENGTH

4.1 Octet Length = 2

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of two octets. The unit of time for this type is 10 ms.

5.3.1.4.20 BinaryTime4

CLASS: **Data Type**

ATTRIBUTES:

1 Data Type Numeric Identifier = 44

2 Data Type Name = BinaryTime4

3 Format = FIXED LENGTH

4.1 Octet Length = 4

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of four octets. The unit of time for this type is 10 µs.

5.3.1.4.21 BinaryTime5

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	45
2	Data Type Name	=	BinaryTime5
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of four octets. The unit of time for this type is 100 µs.

5.3.1.4.22 BinaryTime6

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	46
2	Data Type Name	=	BinaryTime6
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of four octets. The unit of time for this type is 1 ms.

5.3.1.4.23 BinaryTime7

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	47
2	Data Type Name	=	BinaryTime7
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of six octets. The unit of time for this type is 1 ms.

5.3.1.4.24 BinaryTime8

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	48
2	Data Type Name	=	BinaryTime8
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	6

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of six octets. The unit of time for this type is 10 µs.

5.3.1.4.25 BinaryTime9

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	49
2	Data Type Name	=	BinaryTime9
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	6

This type is a binary number. The most significant bit of the most significant octet is always used as the most significant bit of the binary number; no sign bit is included. This binary time type has a length of six octets. The unit of time for this type is 100 µs.

5.3.1.4.26 TIME

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	TIME
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This IEC 61131-3 type is a two's complement binary number with a length of four octets. The unit of time for this type is 1 ms.

5.3.1.4.27 ITIME

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	ITIME
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This IEC 61131-3 type extension is a two's complement binary number with a length of two octets. The unit of time for this type is 1 ms.

5.3.1.4.28 FTIME

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	FTIME
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This IEC 61131-3 type extension is a two's complement binary number with a length of four octets. The unit of time for this type is 1 µs.

5.3.1.4.29 LTIME

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	LTIME
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This IEC 61131-3 type extension is a two's complement binary number with a length of eight octets. The unit of time for this type is 1 µs.

5.3.1.4.30 NetworkTime

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	58
2	Data Type Name	=	NetworkTime
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This data type is composed of an integer value and of an unsigned value that express the network time.

The first element is an Unsigned32 data type that provides the network time in seconds since 1900.01.01 00:00:00(UTC) for network time greater/equal 1984.01.01 00:00:00 (UTC) and

less than or 2036.07.02 06:28:16(UTC), or in seconds since 2036.07.02 06:28:16(UTC) for Network time greater or equal 2036.07.02 06:28:16(UTC).

The second element is an Unsigned32 data type that provides the fractional portion of seconds in $1/2^{32}$s).

5.3.1.4.31 NetworkTimeDifference

CLASS: **Data Type**
ATTRIBUTES:

1	Data Type Numeric Identifier	=	59
2	Data Type Name	=	NetworkTimeDifference
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This data type is composed of an integer value and of an unsigned value that express the difference in network time. The first element is an Integer32 data type that provides the network time difference in seconds. The second element is an Unsigned32 data type that provides the fractional portion of seconds in $1/2^{32}$s.

5.3.1.5 Enumerated types

5.3.1.5.1 PERSISTDEF

CLASS: **Data Type**
ATTRIBUTES:

2	Data Type Name	=	PERSISTDEF
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 2	

The allowed values are shown in Table 1.

Table 1 – PERSISTDEF

Value
CBAVolatile
CBAPendingPersistent
CBAPersistent

5.3.1.5.2 VARTYPE

CLASS: **Data Type**
ATTRIBUTES:

2	Data Type Name	=	VARTYPE
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 2	

The VARTYPE specifies the data type that governs the interpretation of the data. The allowed values are shown in Table 2.

Table 2 – VARTYPE

Value	Data type
VT_EMPTY	no value
VT_NULL	null value (no valid data)
VT_BOOL	VT_BOOLEAN
VT_I1	char
VT_I2	short
VT_I4	long
VT_UI1	unsigned char
VT_UI2	unsigned short
VT_UI4	unsigned long
VT_R4	float
VT_R8	double
VT_CY	Currency
VT_DATE	date
VT_BSTR	address of a BSTR
VT_SAFEARRAY_BOOL	address of a SAFEARRAY (VT_BOOLEAN)
VT_SAFEARRAY_I1	address of a SAFEARRAY (char)
VT_SAFEARRAY_I2	address of a SAFEARRAY (short)
VT_SAFEARRAY_I4	address of a SAFEARRAY (long)
VT_SAFEARRAY_UI1	address of a SAFEARRAY (unsigned char) specified.
VT_SAFEARRAY_UI2	address of a SAFEARRAY (unsigned short) specified.
VT_SAFEARRAY_UI4	address of a SAFEARRAY (unsigned long) specified.
VT_SAFEARRAY_R4	address of a SAFEARRAY (float)
VT_SAFEARRAY_R8	address of a SAFEARRAY (double)
VT_SAFEARRAY_CY	address of a SAFEARRAY (Currency)
VT_SAFEARRAY_DATE	address of a SAFEARRAY (date)
VT_SAFEARRAY_BSTR	address of a SAFEARRAY (BSTR)
VT_DISPATCH	Interface Pointer to an IDispatch interface
VT_UNKNOWN	Interface Pointer to an IUnknown interface
VT_USERDEFINED	address of an userdefined struct
VT_ERROR	A HRESULT is specified.
NOTE All defined structured data types should have the type code VT_USERDEFINED.	

5.3.1.5.3 ITEMQUALITYDEF

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	ITEMQUALITYDEF
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 1	

This data type contains the status information of the related data. It consists of three portions: Quality, Substatus and Limits. There are four states of quality (Bad - the value is not useful; Uncertain - the quality of the value is less than normal, but the value may still be useful; Good (Non Cascade) - the quality of the value is good, possible alarm conditions may be indicated by the substatus; Good (Cascade) - the value may be used in control), a set of sub-status values for each quality, and four states of the limits (OK - the value is free to move; low

limited (LL) - the value has acceded its low limits, high limited (HL) - the value has acceded its high limits; constant (C) - the value cannot move, no matter what the process does). The allowed values are shown in Table 3.

Table 3 – ITEMQUALITYDEF

Quality	Value	Description
Bad	BadNonSpecific	There is no specific reason why the value is bad. Used for propagation.
	BadNonSpecificLL	and low limited
	BadNonSpecificHL	and high limited
	BadNonSpecificC	and constant
	BadConfigurationError	Set if the value is not useful because there is some other problem with the block, depending on what a specific producer can detect.
	BadConfigurationErrorLL	and low limited
	BadConfigurationErrorHL	and high limited
	BadConfigurationErrorC	and constant
	BadNotConnected	Set if this input is required to be connected and is not connected.
	BadNotConnectedLL	and low limited
	BadNotConnectedHL	and high limited
	BadNotConnectedC	and constant
	BadDeviceFailure	Set if the source of the value is affected by a device failure.
	BadDeviceFailureLL	and low limited
	BadDeviceFailureHL	and high limited
	BadDeviceFailureC	and constant
	BadSensorFailure	Set if the device can determine this condition. The limits define which direction has been exceeded.
	BadSensorFailureLL	and low limited
	BadSensorFailureHL	and high limited
	BadSensorFailureC	and constant
	BadLastKnownValue	Set if this value had been set by communication, which has now failed.
	BadLastKnownValueLL	and low limited
	BadLastKnownValueHL	and high limited
	BadLastKnownValueC	and constant
	BadCommFailure	Set if there has never been any communication with this value since it was last Out of Service.
	BadCommFailureLL	and low limited
	BadCommFailureHL	and high limited
	BadCommFailureC	and constant
	BadOutOfService	The value is not reliable because the block is not being evaluated, and may be under construction by a configuration tool. It is set if the block mode is O/S.
	BadOutOfServiceLL	and low limited
	BadOutOfServiceHL	and high limited
	BadOutOfServiceC	and constant

Quality	Value	Description
Uncertain	UncertainNonSpecific	There is no specific reason why the value is uncertain. Used for propagation.
	UncertainNonSpecificLL	and low limited
	UncertainNonSpecificHL	and high limited
	UncertainNonSpecificC	and constant
	UncertainLastUsableValue	Whatever was writing this value has stopped doing so. This is used for fail safe handling.
	UncertainLastUsableValueLL	and low limited
	UncertainLastUsableValueHL	and high limited
	UncertainLastUsableValueC	and constant
	UncertainSubstituteSet	Predefined value is used instead of the calculated one. This is used for fail safe handling.
	UncertainSubstituteSetLL	and low limited
	UncertainSubstituteSetHL	and high limited
	UncertainSubstituteSetC	and constant
	UncertainInitialValue	Value of volatile parameters during and after the reset of the device or a parameter.
	UncertainInitialValueLL	and low limited
	UncertainInitialValueHL	and high limited
	UncertainInitialValueC	and constant
	UncertainSensorNotAccurate	Set if the value is at one of the sensor limits. The limits define which direction has been exceeded. Also set if the device can determine that the sensor has reduced accuracy (e.g. degraded analyzer), in which case no limits are set.
	UncertainSensorNotAccurateLL	and low limited
	UncertainSensorNotAccurateHL	and high limited
	UncertainSensorNotAccurateC	and constant
	UncertainEngineeringUnitsExceeded	Set if the value lies outside of the range of values defined for this parameter. The limits define which direction has been exceeded.
	UncertainEngineeringUnitsExceededLL	and low limited
	UncertainEngineeringUnitsExceededHL	and high limited
	UncertainEngineeringUnitsExceededC	and constant
	UncertainSubNormal	Set if a value derived from multiple values has less than the required number of Good sources.
	UncertainSubNormalLL	and low limited
	UncertainSubNormalHL	and high limited
	UncertainSubNormalC	and constant
	UncertainConfigurationError	Set if there is some inconsistency regarding the parameterization or configuration, depending on what a specific producer can detect.
	UncertainConfigurationErrorLL	and low limited
	UncertainConfigurationErrorHL	and high limited
	UncertainConfigurationErrorC	and constant
	UncertainSimulatedValue	Set when the process value is written by the operator while the block is in manual mode.
	UncertainSimulatedValueLL	and low limited
	UncertainSimulatedValueHL	and high limited
	UncertainSimulatedValueC	and constant

Quality	Value	Description
	UncertainSensorCalibration	Set during the active calibration process together with the current measured value.
	UncertainSensorCalibrationLL	and low limited
	UncertainSensorCalibrationHL	and high limited
	UncertainSensorCalibrationC	and constant
Good	GoodNonCascOk	No error or special condition is associated with this value.
Non Cascade	GoodNonCascOkC	and constant
	GoodNonCascActiveUpdateEvent	Set if the value is good and the block has an active Update Event.
	GoodNonCascActiveUpdateEventLL	and low limited
	GoodNonCascActiveUpdateEventHL	and high limited
	GoodNonCascActiveUpdateEventC	and constant
	GoodNonCascActiveAdvisoryAlarm	Set if the value is good and the block has an active Alarm with a priority less than 8.
	GoodNonCascActiveAdvisoryAlarmLL	and low limited
	GoodNonCascActiveAdvisoryAlarmHL	and high limited
	GoodNonCascActiveAdvisoryAlarmC	and constant
	GoodNonCascActiveCriticalAlarm	Set if the value is good and the block has an active Alarm with a priority greater than or equal to 8.
	GoodNonCascActiveCriticalAlarmLL	and low limited
	GoodNonCascActiveCriticalAlarmHL	and high limited
	GoodNonCascActiveCriticalAlarmC	and constant
	GoodNonCascUnackUpdateEvent	Set if the value is good and the block has an unacknowledged Update Event.
	GoodNonCascUnackUpdateEventLL	and low limited
	GoodNonCascUnackUpdateEventHL	and high limited
	GoodNonCascUnackUpdateEventC	and constant
	GoodNonCascUnackAdvisoryAlarm	Set if the value is good and the block has an unacknowledged Alarm with a priority less than 8.
	GoodNonCascUnackAdvisoryAlarmLL	and low limited
	GoodNonCascUnackAdvisoryAlarmHL	and high limited
	GoodNonCascUnackAdvisoryAlarmC	and constant
	GoodNonCascUnackCriticalAlarm	Set if the value is good and the block has an unacknowledged Alarm with a priority greater than or equal to 8.
	GoodNonCascUnackCriticalAlarmLL	and low limited
	GoodNonCascUnackCriticalAlarmHL	and high limited
	GoodNonCascUnackCriticalAlarmC	and constant
	GoodNonCascInitialFailSafe	This value is from a block that wants its following output block (e.g. AO) to go to Fail Safe.
	GoodNonCascInitialFailSafeLL	and low limited
	GoodNonCascInitialFailSafeHL	and high limited
	GoodNonCascInitialFailSafeC	and constant
	GoodNonCascMaintenanceRequired	The device works still without failure but service support will be necessary soon. This may be detected e.g. by a Transducer Block of a value of pH meter.
	GoodNonCascMaintenanceRequiredLL	and low limited
	GoodNonCascMaintenanceRequiredHL	and high limited
	GoodNonCascMaintenanceRequiredC	and constant

Quality	Value	Description
Good	GoodCascOk	No error or special condition is associated with this value.
Cascade	GoodCascOkC	and constant
	GoodCascInitializationAcknowledge	The value is an initialized value from a source (cascade input, remote-cascade in, and remote-output in parameters).
	GoodCascInitializationAcknowledgeLL	and low limited
	GoodCascInitializationAcknowledgeHL	and high limited
	GoodCascInitializationAcknowledgeC	and constant
	GoodCascInitializationRequest	The value is an initialization value for a source (back calculation input parameter), because the lower loop is broken or the mode is wrong.
	GoodCascInitializationRequestLL	and low limited
	GoodCascInitializationRequestHL	and high limited
	GoodCascInitializationRequestC	and constant
	GoodCascNotInvited	The value is from a block which does not have a target mode that would use this input. This covers all cases other than Fail Safe Active, Local Override, and Not Selected. The target mode can be the next permitted mode of higher priority in the case of shedding a supervisory computer.
	GoodCascNotInvitedLL	and low limited
	GoodCascNotInvitedHL	and high limited
	GoodCascNotInvitedC	and constant
	GoodCascDoNotSelect	The value is from a block which should not be selected, due to conditions in or above the block.
	GoodCascDoNotSelectLL	and low limited
	GoodCascDoNotSelectHL	and high limited
	GoodCascDoNotSelectC	and constant
	GoodCascLocalOverride	The value is from a block that has been locked out by a local key switch or is a Complex AO/DO with interlock logic active. The failure of normal control should be propagated to a PID block for alarm and display purposes. This also implies Not Invited.
	GoodCascLocalOverrideLL	and low limited
	GoodCascLocalOverrideHL	and high limited
	GoodCascLocalOverrideC	and constant
	GoodCascInitiateFailSafe	The value is from a block that wants its downstream output block (e.g. AO) to go to Fail Safe. This is determined by a block option to initiate Fail Safe if the status of the primary input and/or cascade input goes Bad.
	GoodCascInitiateFailSafeLL	and low limited
	GoodCascInitiateFailSafeHL	and high limited
	GoodCascInitiateFailSafeC	and constant

5.3.1.5.4 STATEDEF

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	STATEDEF
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 2	

The allowed values are shown in Table 4.

Table 4 – STATEDEF

Value
CBANonExistent
CBAInitializing
CBAReady
CBAOperating
CBADefect

5.3.1.5.5 GROUPERRORDEF

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	GROUPERRORDEF
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 2	

The allowed values are shown in Table 5.

Table 5 – GROUPERRORDEF

Value
CBANonAccessible
CBAOkay
CBAProblem
CBAUnknown

5.3.1.5.6 ACCESSRIGHTSDEF

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	ACCESSRIGHTSDEF
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 2	

The allowed values are shown in Table 6.

Table 6 – ACCESSRIGHTSDEF

Value
CBANoAccess
CBAReadAccess
CBAWriteAccess
CBAFullAccess

5.3.1.6 Handle types

5.3.1.6.1 HRESULT

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	HRESULT
3	Format	=	FIXED LENGTH
4.1	Octet Length	= 4	

The allowed values are shown in Table 7. The defined HRESULT values may be extended by user specific values.

Table 7 – HRESULT

Value	Description
CBA_E_MALFORMED	The identifier is malformed (too long, unallowed characters, no separation characters, syntactically invalid)
CBA_E_UNKNOWNOBJECT	The object specified in the item identifier is not known.
CBA_E_UNKNOWNMEMBER	The member specified in the item identifier is not known.
CBA_E_TYPEMISMATCH	The type specified does not match the expected type.
CBA_E_INVALIDENUMVALUE	The enumeration value is invalid.
CBA_E_INVALIDID	The ConsumerID or ProviderID is not valid.
CBA_E_INVALIDEPSILON	The Epsilon type or value is not valid.
CBA_E_INVALIDSUBSTITUTE	The Substitute type or value is not valid.
CBA_E_INVALIDCONNECTION	It is not allowed to specify a connection from an identifier to itself.
CBA_E_INVALIDCOOKIE	The Cookie value is not valid.
CBA_E_TIMEVALUEUNSUPPORTED	The time value is unsupported.
CBA_E_QOSTYPEUNSUPPORTED	The QoS type is unsupported.
CBA_E_QOSVALUEUNSUPPORTED	The QoS value is unsupported.
CBA_E_PERSISTRUNNING	While the Save service is running, no changes are allowed to the configuration data base (try again in a few seconds).
CBA_E_INUSE	The destination specified in the item identifier is already connected to some other provider.
CBA_E_NOTAPPLICABLE	The operation is currently not applicable.
CBA_E_NONACCESSIBLE	The item is not accessible.
CBA_E_DEFECT	Hardware defect detected, replacement needed.
CBA_S_PERSISTPENDING	The value requested is currently not persistent.
CBA_S_ESTABLISHING	The connection is not yet established.
CBA_S_NOCONNECTION	There is no connection from the provider to this specific consumer.
CBA_S_VALUEBUFFERED	The value was only buffered and has no immediate effect on the process (e.g. device is in state CBAReady).
CBA_S_VALUEUNCERTAIN	The value requested is currently uncertain.
E_OUTOFMEMORY	Insufficient memory to complete the call.
E_INVALIDARG	One ore more arguments are invalid.
E_NOTIMPL	The service is not implemented.
E_FAIL	Unspecified error.
E_NOINTERFACE	No such interface supported.
RPC_S_PROCNUM_OUT_OF_RANGE	The procedure number is out of range.
RPC_E_INVALID_OXID	The object exporter was not found.
RPC_E_INVALID_OID	The specified object was not found or recognized.
RPC_E_INVALID_SET	The object exporter set was not found.
RPC_E_INVALID_OBJECT	The requested object does not exist.

Value	Description
RPC_E_VERSION_MISMATCH	The ORPC version on the client and server machine does not match.
DISP_E_BADINDEX	Invalid index.
DISP_E_BADPARAMCOUNT	The number of elements provided to DISPPARAMS is different from the number of arguments accepted by the method or property.
DISP_E_BADVARTYPE	One of the arguments is not a valid variant type.
DISP_E_EXCEPTION	The application needs to raise an exception. In this case, the structure passed in pExcepInfo should be filled in.
DISP_E_MEMBERNOTFOUND	The requested member does not exist, or the call to Invoke tried to set the value of a read-only property.
DISP_E_NONAMEDARGS	This implementation of IDispatch does not support named arguments
DISP_E_OVERFLOW	One of the arguments could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	One of the parameter DISPIDs does not correspond to a parameter on the method. In this case, puArgErr should be set to the first argument that contains the error.
DISP_E_TYPEMISMATCH	One or more of the arguments could not be coerced. The index within rgvarg of the first parameter with the incorrect type is returned in the puArgErr parameter.
DISP_E_UNKNOWNINTERFACE	The interface identifier passed in riid is not IID_NULL.
DISP_E_UNKNOWNLCID	Unknown language.
DISP_E_UNKNOWNNAME	Unknown name.
DISP_E_PARAMNOTOPTIONAL	A required parameter was omitted.
TYPE_E_ELEMENTNOTFOUND	Element not found.
S_OK	Success, "everything worked".
S_FALSE	Success but with additional results, "function worked and the result is false".

5.3.1.7 Numeric types

5.3.1.7.1 BCD

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	35
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 octet 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. In this type, the least significant four bits are used to express a BCD value which is between zero and nine inclusive. The most significant four bits are unused.

5.3.1.7.2 Floating Point types

5.3.1.7.2.1 Float32

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	8
2	Data Type Name	=	Float32
4	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type has a length of four octets. The format for float32 is that defined by IEC 60559 as single precision.

5.3.1.7.2.2 Floating point

This data type is the same as Float32.

5.3.1.7.2.3 REAL

This IEC 61131-3 type is the same as Float32.

5.3.1.7.2.4 float

This data type is the same as Float32.

5.3.1.7.2.5 Float64

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	15
2	Data Type Name	=	Float64
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This type has a length of eight octets. The format for float64 is that defined by IEC 60559 as double precision.

5.3.1.7.2.6 LREAL

This IEC 61131-3 type is the same as Float64.

5.3.1.7.2.7 double

This data type is the same as Float64.

5.3.1.7.3 Integer types

5.3.1.7.3.1 Integer8

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	2
2	Data Type Name	=	Integer8
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	1

This integer type is a two's complement binary number with a length of one octet.

5.3.1.7.3.2 SINT

This IEC 61131-3 type is the same as Integer8.

5.3.1.7.3.3 char

This data type is the same as Integer8.

5.3.1.7.3.4 Integer16

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	3
2	Data Type Name	=	Integer16
3	Format	=	FIXED LENGTH

4.1 Octet Length = 2

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

5.3.1.7.3.5 INT

This IEC 61131-3 type is the same as Integer16.

5.3.1.7.3.6 short

This data type is the same as Integer16.

5.3.1.7.3.7 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.

5.3.1.7.3.8 DINT

This IEC 61131-3 type is the same as Integer32.

5.3.1.7.3.9 long

This data type is the same as Integer32.

5.3.1.7.3.10 Integer64

CLASS: **Data Type**

ATTRIBUTES:

1 Data Type Numeric Identifier = 55

2 Data Type Name = Integer64

3 Format = FIXED LENGTH

4.1 Octet Length = 8

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

5.3.1.7.3.11 LINT

This IEC 61131-3 type is the same as Integer64.

5.3.1.7.4 Unsigned types

5.3.1.7.4.1 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 octet 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.

5.3.1.7.4.2 USINT

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

5.3.1.7.4.3 unsigned char

This data type is the same as Unsigned8.

5.3.1.7.4.4 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 octet 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.

5.3.1.7.4.5 UINT

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

5.3.1.7.4.6 unsigned short

This data type is the same as Unsigned16.

5.3.1.7.4.7 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 octet 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.

5.3.1.7.4.8 UDINT

This IEC 61131-3 type is the same as Unsigned32.

5.3.1.7.4.9 unsigned long

This data type is the same as Unsigned32.

5.3.1.7.4.10 Unsigned64

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	56
2	Data Type Name	=	Unsigned64
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

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

5.3.1.7.4.11 ULINT

This IEC 61131-3 type is the same as Unsigned64.

5.3.1.8 Pointer types

5.3.1.8.1 Interface pointer

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	Interface Pointer
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This data type defines a 4 octet fixed length data type.

5.3.1.8.2 LPWSTR

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	LPWSTR
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This data type defines a reference to an UnicodeString.

5.3.1.9 OctetString character types

5.3.1.9.1 OctetString1

CLASS: Data Type
ATTRIBUTES:

1	Data Type Numeric Identifier	=	30
2	Data Type Name	=	OctetString1
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	1

This type has a length of one octet.

5.3.1.9.2 OctetString2

CLASS: Data Type
ATTRIBUTES:

1	Data Type Numeric Identifier	=	31
2	Data Type Name	=	OctetString2
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This type has a length of two octets.

5.3.1.9.3 OctetString4

CLASS: Data Type
ATTRIBUTES:

1	Data Type Numeric Identifier	=	32
2	Data Type Name	=	OctetString4
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type has a length of four octets.

5.3.1.9.4 OctetString8

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	33
2	Data Type Name	=	OctetString8
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This octet string type has a length of eight octets.

5.3.1.9.5 OctetString16

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	34
2	Data Type Name	=	OctetString16
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	16

This type has a length of 16 octets.

5.3.1.9.6 UUID

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	UUID
3	Format	=	STRUCTURE
5.1	Number of Fields	=	4
5.2.1	Field Name	=	Data1
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	Data2
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	Data3
5.2.6	Field Data Type	=	unsigned short
5.2.7	Field Name	=	Data4
5.2.8.1	Format	=	ARRAY
5.2.8.4.1	Number of Array Elements	=	8
5.2.8.4.2	Array Element Data Type	=	unsigned char

This data type defines a 16 octet fixed length data type. The semantic is specified by the used ORPC model and beyond the scope of this standard.

This data type is structured as follows:

Data1
This field contains the first eight hexadecimal digits of the UUID;

Data2
This field contains the first group of four hexadecimal digits of the UUID;

Data3
This field contains the second group of four hexadecimal digits of the UUID;

Data4
This field contains an array of eight elements. The first two elements contain the third group of four hexadecimal digits of the UUID. the remaining six elements contain the final 12 hexadecimal digits of the UUID.

Predefined values for TYPE 10 items are shown in Table 8.

Table 8 – UUID

Value	Description
UUID_NULL	
UUID_IUnknown	Identifies the IUnknown interface uniquely.
UUID_IDispatch	Identifies the IDispatch interface uniquely.
UUID_ICBAPhysicalDevice	Identifies the ICBAPhysicalDevice interface uniquely.
UUID_ICBABrowse	Identifies the ICBABrowse interface uniquely.
UUID_ICBAPersist	Identifies the ICBAPersist interface uniquely.
UUID_ICBALogicalDevice	Identifies the ICBALogicalDevice interface uniquely.
UUID_ICBAState	Identifies the ICBAState interface uniquely.
UUID_ICBATime	Identifies the ICBATime interface uniquely.
UUID_ICBAGroupError	Identifies the ICBAGroupError interface uniquely.
UUID_ICBAAccoMgt	Identifies the ICBAAccoMgt interface uniquely.
UUID_ICBAAccoServer	Identifies the ICBAAccoServer interface uniquely.
UUID_ICBAAccoCallback	Identifies the ICBAAccoCallback interface uniquely.
UUID_ICBAAccoSync	Identifies the ICBAAccoSync interface uniquely.
UUID_ICBARTAuto	Identifies the ICBARTAuto interface uniquely.
UUID_PhysicalDevice	Identifies the Physical Device class uniquely.
UUID_LogicalDevice	Identifies the Logical Device class uniquely.
UUID_ACCO	Identifies the ACCO class uniquely.
UUID_RTAuto	Identifies the RTAuto class uniquely.

5.3.1.10 VisibleString character types

5.3.1.10.1 UNICODE char

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	36
2	Data Type Name	=	UnicodeChar
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This type is defined as a single character in the UNICODE string type.

5.3.1.10.2 VisibleString1

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	25
2	Data Type Name	=	VisibleString1
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	1

This type is defined as a single character in the ISO VisibleString type.

5.3.1.10.3 VisibleString2

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	26
2	Data Type Name	=	VisibleString2

3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This type contains two elements of type VisibleString.

5.3.1.10.4 VisibleString4

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	27
2	Data Type Name	=	VisibleString4
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type contains four elements of type VisibleString.

5.3.1.10.5 VisibleString8

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	28
2	Data Type Name	=	VisibleString8
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	8

This type contains eight elements of type VisibleString.

5.3.1.10.6 VisibleString16

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	29
2	Data Type Name	=	VisibleString16
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	16

This type contains 16 elements of type VisibleString.

5.3.2 String types

5.3.2.1 BitString

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	14
2	Data Type Name	=	Bitstring
3	Format	=	STRING
5.1	Octet Length	=	1 to n

This string type is defined as a series of BitString8 elements.

5.3.2.2 CompactBooleanArray

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	37
2	Data Type Name	=	CompactBooleanArray
3	Format	=	STRING
6.1	Octet Length	=	1

In this type, each bit value of 0 (zero) represents the Boolean value FALSE and each bit value of 1 represents the Boolean value TRUE.

5.3.2.3 CompactBCDArray

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	38
2	Data Type Name	=	CompactBCDArray
3	Format	=	STRING
4.1	Octet Length	=	1

This type is used to pack an ordered series of BCD values, two per octet, into an ordered series of octets. The first octet contains the most significant BCD value in its most significant quartet. If the number of BCD values is odd, the least significant quartet of the final octet is set to the reserved value "1111".

5.3.2.4 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 61158-6-11 defines the order of transmission.

5.3.2.5 UNICODEString

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	39
2	Data Type Name	=	UnicodeString
3	Format	=	STRING
4.1	Octet Length	=	2

This type is defined as the UNICODE string type.

5.3.2.6 VisibleString

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	9
2	Data Type Name	=	VisibleString
3	Format	=	STRING
4.1	Octet Length	=	1 to n

This type is defined as the ISO/IEC 646 string type.

5.3.3 Structure types

5.3.3.1 ADDCONNECTIONIN

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	ADDCONNECTIONIN
3	Format	=	STRUCTURE
5.1	Number of Fields	=	5

5.2.1	Field Name	=	ProviderItem
5.2.2	Field Data Type	=	LPWSTR
5.2.3	Field Name	=	ConsumerItem
5.2.4	Field Data Type	=	LPWSTR
5.2.5	Field Name	=	Persistence
5.2.6	Field Data Type	=	PERSISTDEF
5.2.7	Field Name	=	SubstituteValue
5.2.8	Field Data Type	=	VARIANT
5.2.9	Field Name	=	Epsilon
5.2.10	Field Data Type	=	VARIANT

This data type defines the ADDCONNECTIONIN data type.

This data type is structured as follows:

ProviderItem
This field contains the name of the source data item;

ConsumerItem
This field contains the name of the sink data item;

Persistence
This field describes the required persistence of the connection information. The values CBAVolatile and CBAPersistent are allowed;

SubstituteValue
This field specifies the substitute value according the data type of the connection item;

Epsilon
This field specifies the hysteresis as absolute change of the value.

5.3.3.2 ADDCONNECTIONOUT

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	ADDCONNECTIONOUT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	3
5.2.1	Field Name	=	ConsumerID
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	Version
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	ErrorState
5.2.6	Field Data Type	=	HRESULT

This data type defines the ADDCONNECTIONOUT data type.

This data type is structured as follows:

ConsumerID
This field contains the identifier of the consumer;

Version
This field contains the version number of the connection;

ErrorState
This field contains the error condition of the connection.

5.3.3.3 BSTR

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	BSTR
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	OctetCount

5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	UnicodeString
5.2.4	Field Data Type	=	UnicodeString

This data type defines an UnicodeString with preceding octet count value. The count contains the number of octets, not UNICODE characters, in the string. This count does not include the terminating zero character (2 octets).

5.3.3.4 CONNECTIN

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	CONNECTIN
3	Format	=	STRUCTURE
5.1	Number of Fields	=	4
5.2	List of Fields		
5.2.1	Field Name	=	ProviderItem
5.2.2	Field Data Type	=	LPWSTR
5.2.3	Field Name	=	DataType
5.2.4	Field Data Type	=	VARTYPE
5.2.5	Field Name	=	Epsilon
5.2.6	Field Data Type	=	VARIANT
5.2.7	Field Name	=	ConsumerID
5.2.8	Field Data Type	=	unsigned long

This data type defines the CONNECTIN data type.

This data type is structured as follows:

ProviderItem
This field contains the name of the source data item;

DataType
This field contains the type code for the data item;

Epsilon
This field specifies the hysteresis as absolute change of the value;

ConsumerID
This field contains the identifier of the consumer.

5.3.3.5 CONNECTOUT

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	CONNECTOUT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	ProviderID
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	ErrorState
5.2.4	Field Data Type	=	HRESULT

This data type defines the CONNECTOUT data type.

This data type is structured as follows:

ProviderID
This field contains the identifier of the provider;

ErrorState
This field contains the error condition of the interconnection.

5.3.3.6 EXCEPTINFO

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	EXCEPINFO
3	Format	=	STRUCTURE
5.1	Number of Fields	=	9
5.2.1	Field Name	=	`wCode`
5.2.2	Field Data Type	=	unsigned short
5.2.3	Field Name	=	wReserved
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	bstrSource
5.2.6	Field Data Type	=	BSTR
5.2.7	Field Name	=	bstrDescription
5.2.8	Field Data Type	=	BSTR
5.2.9	Field Name	=	bstrHelpFile
5.2.10	Field Data Type	=	BSTR
5.2.11	Field Name	=	dwHelpContext
5.2.12	Field Data Type	=	unsigned long
5.2.13	Field Name	=	pvReserved
5.2.14	Field Data Type	=	unsigned long
5.2.15	Field Name	=	pfnDeferredFillIn
5.2.16	Field Data Type	=	unsigned long
5.2.17	Field Name	=	scode
5.2.18	Field Data Type	=	long

This data type defines the EXCEPINFO data type to describe an exception that occurred during the Invoke service.

This data type is structured as follows:

wCode
This field contains an error code identifying the error. Error codes have a value greater than 1000. Either this field or the scode field, but not both, should have a non-zero value; the other field should have the value 0.

wReserved
This field is reserved and should be set to 0;

bstrSource
This field contains a textual, human-readable name of the source of the exception. Typically, this is an application name. This field should be provided by the implementor of the IDispatch interface;

bstrDescription
This field contains a textual, human-readable description of the error intended for the customer. If no description is available, use Null;

bstrHelpFile
This field contains the fully qualified drive, path, and file name of a Help file with more information about the error. If no Help is available, use Null;

dwHelpContext
This field contains the Help context ID of the topic within the Help file. This field is relevant if and only if the bstrHelpFile field is not Null;

pvReserved
This field is reserved and should be set to Null;

pfnDeferredFillIn
This field contains the address of a function that takes an EXCEPINFO structure as an argument and returns an HRESULT value. If deferred, fill-in is not desired, this field should be set to Null;

scode
This field contains a return value describing the error. Either this field or the wCode field, but not both, should have a non-zero value; the other field should have the value 0.

5.3.3.7 DATE_AND_TIME

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	DATE_AND_TIME
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2.1	Field Name	=	Time_Of_Day_Element
5.2.2	Field Data Type	=	TIME_OF_DAY
5.3.1	Field Name	=	Date_Element
5.3.2	Field Data Type	=	DATE

This IEC 61131-3 type extension is a structure which expresses both the date (as a number of days starting from January 1st, 1972 until June 6th, 2151),and the time of day as a number of ms starting from midnight.

5.3.3.8 FILETIME

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	FILETIME
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	LowDataTime
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	HighDataTime
5.2.4	Field Data Type	=	unsigned long

This data type is a 64 bit time value representing the number of 100-nanosecond intervals since January 1, 1601.

5.3.3.9 GETIDOUT

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	GETIDOUT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	4
5.2.1	Field Name	=	ConsumerID
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	State
5.2.4	Field Data Type	=	VT_BOOLEAN
5.2.5	Field Name	=	Version
5.2.6	Field Data Type	=	unsigned short
5.2.7	Field Name	=	ErrorState
5.2.8	Field Data Type	=	HRESULT

This data type defines the GETIDOUT data type.

This data type is structured as follows:

ConsumerID
This field contains the identifier of the consumer;

State
This field contains the state of the connection. The value TRUE indicates an active and the value FALSE an inactive connection;

Version
This field contains the version number of the connection;

ErrorState
This field contains the error condition of the connection.

5.3.3.10 GETCONNECTIONOUT

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	GETCONNECTIONOUT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	10
5.2.1	Field Name	=	Provider
5.2.2	Field Data Type	=	LPWSTR
5.2.3	Field Name	=	ProviderItem
5.2.4	Field Data Type	=	LPWSTR
5.2.5	Field Name	=	ConsumerItem
5.2.6	Field Data Type	=	LPWSTR
5.2.7	Field Name	=	SubstituteValue
5.2.8	Field Data Type	=	VARIANT
5.2.9	Field Name	=	Epsilon
5.2.10	Field Data Type	=	VARIANT
5.2.11	Field Name	=	QoSType
5.2.12	Field Data Type	=	unsigned short
5.2.13	Field Name	=	QoSValue
5.2.14	Field Data Type	=	unsigned short
5.2.15	Field Name	=	State
5.2.16	Field Data Type	=	VT_BOOLEAN
5.2.17	Field Name	=	Persistence
5.2.18	Field Data Type	=	PERSISTDEF
5.2.19	Field Name	=	Version
5.2.20	Field Data Type	=	unsigned short
5.2.21	Field Name	=	ErrorState
5.2.22	Field Data Type	=	HRESULT

This data type defines the GETCONNECTIONOUT data type.

This data type is structured as follows:

Provider
This field contains the name of the providers LDev (Format: PDev!LDev);

ProviderItem
This field contains the name of the source data item;

ConsumerItem
This field contains the name of the sink data item;

SubstituteValue
This field specifies the substitute value according the data type of the connection item;

Epsilon
This field specifies the hysteresis as absolute change of the value;

QoSType
This field contains the quality of service type;

QoSValue
This field contains the quality of service qualifier;

State
This field contains the state of the connection. The value TRUE indicates an active and the value FALSE an inactive connection;

Persistence
This field describes the required persistence of the connection information. The values CBAVolatile and CBAPersistent are allowed;

Version
This field contains the version number of the connection;

ErrorState
This field contains the error condition of the connection.

5.3.3.11 QualifiedOctetString2

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	QualifiedOctetString2
5	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	Value
5.2.2	Field Data Type	=	Octet String(2)
5.2.3	Field Name	=	Status
5.2.4	Field Data Type	=	Unsigned8

This data type defines the QualifiedOctetString2.

This data type is structured as follows.

Value
This field contains the value as Octet String with the length of 2.

Status
This field describes the status of the value as a qualifier.

5.3.3.12 QualifiedFloat32

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	QualifiedFloat32
5	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	Value
5.2.2	Field Data Type	=	Float32
5.2.3	Field Name	=	Status
5.2.4	Field Data Type	=	Unsigned8

This data type defines the QualifiedFloat32.

This data type is structured as follows.

Value
This field contains the value as Float32.

Status
This field describes the status of the value as a qualifier.

5.3.3.13 QualifiedUnsigned8

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	QualifiedUnsigned8
5	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	Value
5.2.2	Field Data Type	=	Unsigned8
5.2.3	Field Name	=	Status

5.2.4 Field Data Type = Unsigned8

This data type defines the QualifiedUnsigned8.

This data type is structured as follows.

Value
This field contains the value as Unsigned8.

Status
This field describes the status of the value as a qualifier.

5.3.3.14 READITEMOUT

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	READITEMOUT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	4
5.2.1	Field Name	=	Value
5.3.2	Field Data Type	=	VARIANT
5.4.3	Field Name	=	QualityCode
5.5.4	Field Data Type	=	ITEMQUALITYDEF
5.6.5	Field Name	=	TimeStamp
5.7.6	Field Data Type	=	FILETIME
5.8.7	Field Name	=	ErrorState
5.9.8	Field Data Type	=	HRESULT

This data type defines the READITEMOUT data type.

This data type is structured as follows:

Value
This field contains the value itself with the format according to the appropriate data type;

QualityCode
This field contains the Quality Code of the data value;

TimeStamp
This field contains the time stamp of the value;

ErrorState
This field contains the error condition of the connection.

5.3.3.15 SAFEARRAY

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	RGSABOUND
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2	List of Fields		
5.2.1	Field Name	=	Elements
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	Left Bound
5.2.4	Field Data Type	=	long

This data type is a helper describing one array element with boundary information needed to define the SAFEARRAY below.

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	SAFEARRAY
3	Format	=	STRUCTURE
5.1	Number of Fields	=	6
5.2	List of Fields		
5.2.1	Field Name	=	Dims
5.2.2	Field Data Type	=	unsigned short

5.2.3	Field Name	=	Features
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	Elements
5.2.6	Field Data Type	=	unsigned long
5.2.7	Field Name	=	Locks
5.2.8	Field Data Type	=	unsigned long
5.2.9	Field Name	=	Data Address
5.2.10	Field Data Type	=	unsigned long
5.2.11	Field Name	=	Rgsabound
5.2.12	Format	=	ARRAY
5.2.12.1	Number of Array Elements	=	Dims
5.2.12.2	Array Element Data Type	=	RGSABOUND

This data type expresses a one- or multi-dimensional array of a single data type. (However, this single data type can be a VARIANT, allowing you arrays of mixed types.) The reason these arrays are called safe arrays is because they contain bounds information, allowing to check the index or indices against the bounds before accessing the data in the array. The lower bound of the array does not have to be zero, so the safe array has to store its lower bound as well as the size. Finally, safe arrays allow locking (and unlocking) so it can be sure the pointer to the data you get is valid.

This data type is structured as follows:

Dims
This field contains the dimension of the SAFEARRAY;

Features
This field contains flags for allocation type and data type of the SAFEARRAY;

Elements
This field contains the size of a single element of the SAFEARRAY;

Locks
This field contains the lock counter of the SAFEARRAY;

Data Address
This field contains the address of the actual data of the SAFEARRAY;

Rgsabound
This field contains an array with the information about boundaries for each dimension of the SAFEARRAY. Therefore, the number of array elements is according to the dimension of the safe array

5.3.3.16 SHORT_STRING

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	SHORT_STRING
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2.1	Field Name	=	Charcount_Element
5.2.2	Field Data Type	=	USINT
5.3.1	Field Name	=	Stringcontents_Element
5.3.2	Field Data Type	=	OctetString

This IEC 61131-3 type extension is composed of two elements. Charcount_Element gives the current number of characters in the Stringcontents_Element (one USINT per character).

5.3.3.17 STRING

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	STRING
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2.1	Field Name	=	Charcount_Element
5.2.2	Field Data Type	=	UINT
5.3.1	Field Name	=	Stringcontents_Element
5.3.2	Field Data Type	=	OctetString

This IEC 61131-3 type is composed of two elements. Charcount_Element gives the current number of characters in the Stringcontents_Element (one USINT per character).

5.3.3.18 STRING2

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	STRING2
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2.1	Field Name	=	Charcount_Element
5.2.2	Field Data Type	=	UINT
5.3.1	Field Name	=	String2contents_Element
5.3.2	Field Data Type	=	OctetString

This IEC 61131-3 data type extension is composed of two elements. Charcount_Element gives the current number of characters in the String2contents_Element (one UINT per character).

5.3.3.19 STRINGN

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	not used
2	Data Type Name	=	STRINGN
3	Format	=	STRUCTURE
5.1	Number of Fields	=	3
5.2.1	Field Name	=	Charsize_Element
5.2.2	Field Data Type	=	UINT
5.3.1	Field Name	=	Charcount_Element
5.3.2	Field Data Type	=	UINT
5.4.1	Field Name	=	StringNcontents_Element
5.4.2	Field Data Type	=	OctetString

This IEC 61131-3 type extension is composed of three elements. Charsize_Element gives the size of a character in StringNcontents_Element (N = number of USINT). Charcount_Element gives the current number of characters in the StringNcontents_Element (N USINT per character).

5.3.3.20 VARIANT

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	VARIANT
3	Format	=	STRUCTURE
5.1	Number of Fields	=	5

5.2	List of Fields		
5.2.1	Field Name	=	Tag
5.2.2	Field Data Type	=	VARTYPE
5.2.3	Field Name	=	Padding1
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	Padding2
5.2.6	Field Data Type	=	unsigned short
5.2.7	Field Name	=	Padding3
5.2.8	Field Data Type	=	unsigned short
5.2.9	Field Name	=	Value
5.2.10	Field Data Type	=	see Table 9

This data type expresses a VARIANT that containing possible values of data types according to Table 9. The Tag field contains the numeric identifier of the data type to identify the data type of the value. The overall size of the VARIANT is 16 octets.

Table 9 – Data type names for value

Data Type
VT_BOOLEAN
char
short
long
unsigned char
unsigned short
unsigned long
float
double
date
currency
Address of a BSTR
Address of a SAFEARRAY
Address of a userdefined struct
Interface Pointer
HRESULT

5.3.3.21 WRITEITEMIN

CLASS:	**Data Type**		
ATTRIBUTES:			
2	Data Type Name	=	WRITEITEMIN
3	Format	=	STRUCTURE
5.1	Number of Fields	=	2
5.2.1	Field Name	=	Item
5.2.2	Field Data Type	=	LPWSTR
5.2.3	Field Name	=	Value
5.2.4	Field Data Type	=	VARIANT

This data type defines the WRITEITEMIN data type.

This data type is structured as follows:

Item
This field contains the name of the data item;

Value
This field contains the value itself with the format according to the appropriate data type.

5.3.3.22 WRITEITEMQCDIN

CLASS: Data Type
ATTRIBUTES:

2	Data Type Name	=	WRITEITEMQCDIN
3	Format	=	STRUCTURE
5.1	Number of Fields	=	3
5.2.1	Field Name	=	WriteItem
5.2.2	Field Data Type	=	WRITEITEMIN
5.2.3	Field Name	=	QualityCode
5.2.4	Field Data Type	=	ITEMQUALITYDEF
5.2.5	Field Name	=	TimeStamp
5.2.6	Field Data Type	=	FILETIME

This data type defines the WRITEITEMQCDIN data type.

This data type is structured as follows:

WriteItem
This field contains the name and value of the data item;

QualityCode
This field contains the Quality Code of the data value;

TimeStamp
This field contains the time stamp of the value.

5.3.3.23 DISPPARAMS

2	Data Type Name	=	UUID
3	Format	=	STRUCTURE
5.1	Number of Fields	=	4
5.2.1	Field Name	=	Data1
5.2.2	Field Data Type	=	unsigned long
5.2.3	Field Name	=	Data2
5.2.4	Field Data Type	=	unsigned short
5.2.5	Field Name	=	Data3
5.2.6	Field Data Type	=	unsigned short
5.2.7	Field Name	=	Data4
5.2.8.1	Format	=	ARRAY
5.2.8.4.1	Number of Array Elements	=	8
5.2.8.4.2	Array Element Data Type	=	unsigned char

This data type defines a 16 octet fixed length data type. The semantic is specified by the used ORPC model and beyond the scope of this standard.

This data type is structured as follows:

Data1
This field contains the first eight hexadecimal digits of the UUID;

Data2
This field contains the first group of four hexadecimal digits of the UUID;

Data3
This field contains the second group of four hexadecimal digits of the UUID;

Data4
This field contains an array of eight elements. The first two elements contain the third group of four hexadecimal digits of the UUID. the remaining six elements contain the final 12 hexadecimal digits of the UUID.

Predefined values for TYPE 10 items are shown in Table 10.

Table 10 – UUID

Value	Description
UUID_NULL	
UUID_IUnknown	Identifies the IUnknown interface uniquely.
UUID_IDispatch	Identifies the IDispatch interface uniquely.
UUID_ICBAPhysicalDevice	Identifies the ICBAPhysicalDevice interface uniquely.
UUID_ICBABrowse	Identifies the ICBABrowse interface uniquely.
UUID_ICBAPersist	Identifies the ICBAPersist interface uniquely.
UUID_ICBALogicalDevice	Identifies the ICBALogicalDevice interface uniquely.
UUID_ICBAState	Identifies the ICBAState interface uniquely.
UUID_ICBATime	Identifies the ICBATime interface uniquely.
UUID_ICBAGroupError	Identifies the ICBAGroupError interface uniquely.
UUID_ICBAAccoMgt	Identifies the ICBAAccoMgt interface uniquely.
UUID_ICBAAccoServer	Identifies the ICBAAccoServer interface uniquely.
UUID_ICBAAccoCallback	Identifies the ICBAAccoCallback interface uniquely.
UUID_ICBAAccoSync	Identifies the ICBAAccoSync interface uniquely.
UUID_ICBARTAuto	Identifies the ICBARTAuto interface uniquely.
UUID_PhysicalDevice	Identifies the Physical Device class uniquely.
UUID_LogicalDevice	Identifies the Logical Device class uniquely.
UUID_ACCO	Identifies the ACCO class uniquely.
UUID_RTAuto	Identifies the RTAuto class uniquely.

5.4 Data type ASE service specification

There are no operational services defined for the type object.

6 Communication model specification

6.1 ASEs

6.1.1 Common memory ASE

6.1.1.1 Overview

The CM ASE provides the Update_Memory service for the ALS-user to read and write data from/to the Common Memory in order to intercommunication between processes running on remote nodes.

6.1.1.2 Common memory model class specification

6.1.1.2.1 Common memory formal model

ASE:			CM ASE
CLASS:			CM
CLASS ID:			
PARENT CLASS:			TOP
ATTRIBUTES:			
1	(m)	Key Attribute:	Not used
2	(m)	Attribute:	Role (Publisher, Subscriber)
3	(m)	Attribute:	State
4	(m)	Attribute:	Common memory
4.1	(m)	Attribute:	Total memory size
4.2	(m)	Attribute:	Block memory size
SERVICES:			
1	(o)	Ops Service	Update-Memory
2	(o)	Ops Service	Memory-Status

6.1.1.2.2 Attributes

Role

This attribute specifies the role for each common memory block with the same meaning as AREP. The valid values are as follows:

Publisher Endpoint of this type publishes their data by issuing Update-Memory service request primitive;

Subscriber Endpoint of this type subscribes the data in response to Update-Memory service indication primitive.

State

This attribute indicates the current state of the CM ASE (FSPM) that is defined in detail in IEC 61158-6-11.

Common memory

The following attributes specify the capacity of the CM.

Total memory size

The capacity of the whole common memory is specified in octets.

Block memory size

The size of the block associated with a AREP is specified in octets.

6.1.1.2.3 Services

Update memory

The Update_Memory request primitive is used by the ALS-user to update the content of a bock of the Common Memory as a publisher.

The Update_Memory indication primitive is used to notify the ALS-user that the designated block is updated.

Memory-status

The Memory Status indication primitive is used to notify the ALS-user the timing to publish.

6.1.1.3 Service specification of common memory (CM) ASE

6.1.1.3.1 Supported services

The services provided by Common Memory ASE are as follows:

Update Memory;

Memory-Status.

6.1.1.3.2 Update memory service

6.1.1.3.2.1 Service overview

The Update_Memory service is used by the ALS-user to update the content of the block of the Common Memory as a publisher. AR is based on the Push model of Publisher and Subscriber.

6.1.1.3.2.2 Service primitives

The service parameters for this service is shown in Table 11.

Table 11 – Update memory service parameters

Parameter name	Req	Ind
Argument		
AREP	M	M (=)
Memory Contents	M	M (=)

Argument

The argument contain the parameters of the Update-Memory service request.

Memory contents

This parameter specifies the Common memory corresponding to the AREP.

6.1.1.3.2.3 Service procedure

The ALS-user issues Update_Memory request primitive to the Common Memory ASE, and the ASE assembles APDU in unacknowledged type and hands over the APDU to the AR ASE.

The ASE on remote node as subscriber informs the ALS-user on reception of the APDU from other remote node using Update_Memory indication primitive.

6.1.1.3.3 Memory-status service

6.1.1.3.3.1 Service overview

Memory Status service informs the ALS-user the status of the local memory identical the global Common Memory.

6.1.1.3.3.2 Service primitives

The service parameters for this service are shown in Table 12.

Table 12 – Memory-status service parameters

Parameter name	Ind
Argument	
AREP	M
Memory status code	M

Argument

This argument contains the parameter corresponding to Memory-Status service.

Memory status code

This parameter indicates the status of the Common Memory.

Now-Updating. -- for use in Publisher side

6.1.1.3.3.3 Service procedure

The ASE issues the Memory_status indication primitive to the ALS-user.

6.1.2 Application relationship ASE

6.1.2.1 Overview

6.1.2.1.1 General

IEC/TR 61158-1 shows the CM Publisher/Subscriber model. The set of AREPs which is commonly used in this standard, is predefined and pre-established in static connection of each other. The relationship between local and remote APs is distinguished by the AREP identifier. In relationship between Publisher and Subscriber associated with AREP identifier, the AR is one to many relationship.

Figure 7 also depicts the data flow on the model of AREP class for Publisher/Subscriber in the CM, that is the buffered network-scheduled uni-directional pre-established connection（BNU-PEC) AREP.

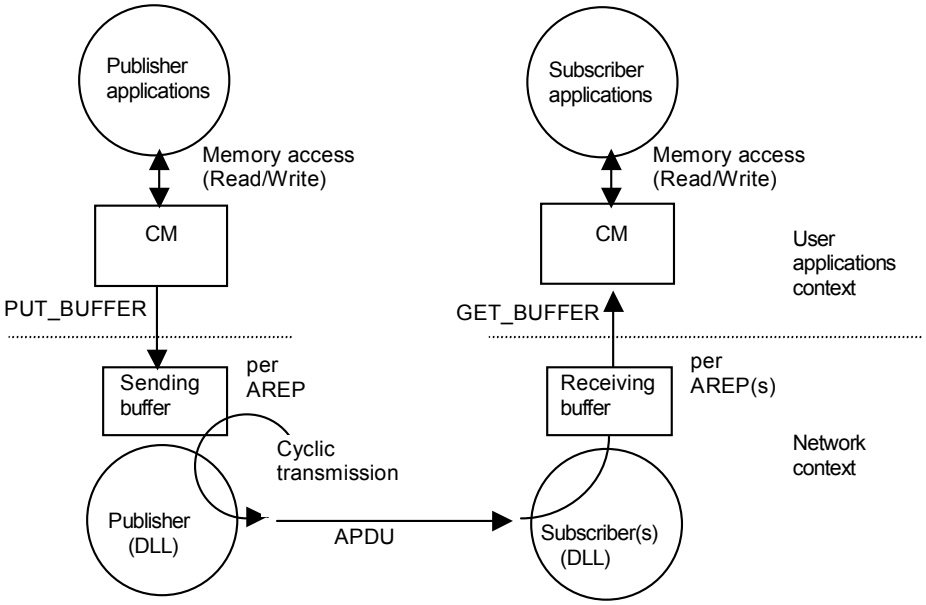

Figure 7 – Common memory publisher/subscriber model

6.1.2.1.2 AR-endpoint

The application on the Common Memory uses the predefined common AREP among each other. For one AREP a single publisher is defined and assigned in RTE-TCnet. If defined duplicated publishers for one AREP, that should be alarmed and notified.

6.1.2.1.3 AR-endpoint class

The AR is established at each AREP common in the RTE-TCnet domain.

6.1.2.1.4 AR cardinality

ARs characterize communications between APs. One of the characteristics of an AR is the number of AREPs in the AR. The buffered network-scheduled uni-directional pre-established connection (BNU-PEC) AREPs convey services from one AP to a number of APs and have a cardinality of one-to-many.

6.1.2.1.5 Accessing object through ARs

The AR provides with the means to access the Common Memory using the services by the ASE.

6.1.2.1.6 AR conveyance paths

The AR is modelled as the communication path with uni-directional data transfer among AREPs. The AREP at the receiver side on the AR receives the data sent by the AREP at the sender side.

6.1.2.1.7 AREP role

The role of AREP is based on the Publisher/Subscriber model with Push type.

6.1.2.1.8 AREP buffers

The AREP is modelled from the viewpoint of the data buffer provided with the DL service and protocol. The APDU to be conveyed over through the BNU-PEC AREP is stored into the Send-buffer of the DL and is sent out by the DLE over the transmission media.

Once the Send-buffer is updated, the APDU is sent out. In that case the APDU is retained in the Send-buffer and is read out from the Send-buffer for cyclic data transmission.

On the contrary at the receiver side the situation is inversive.

The APDU received at the receiver side is stored into the Receive-buffer of the DL. The consecutive APDU received is overwritten to the Receive-buffer of the data-link layer.

The reading out of the Receive-buffer is not in destructive nature and the APDU received is maintained in the Receive-buffer so that the ALS-user can read out over and over.

6.1.2.1.9 Network scheduled conveyance

The buffered network-scheduled uni-directional pre-established connection (BNU-PEC) AREP cyclically issues the request to send out data at the interval specified. The interval is controlled and maintained by the DLE.

6.1.2.1.10 Identification of ARs

The AR is identified and established by the DLCEP-identifier.

6.1.2.1.11 Definitions and establishment of AREPs

The definition of the AREP specifies the instance of an AREP class.

The AREPs are pre-defined and pre-established.

6.1.2.1.12 AR establishment and termination

The AREP is predefined and is identified by the AREP identifier, the assignment of which is maintained until power-off or RESET request by the MLE.

6.1.2.2 Application relationship endpoint class specification

6.1.2.2.1 Formal model

The corresponding ARs carry out Unconfirmed Services (UCS) over a buffer, network scheduled , unidirectional AR which utilizes a pre-established data link connection.

ASE:			**AR ASE**
CLASS:			AR ENDPOINT
CLASS ID:			32
PARENT CLASS:			TOP
ATTRIBUTE:			
1	(m)	Attribute:	Local AP
2	(m)	Attribute:	FAL Revision
3	(m)	Attribute:	Dedicated (TRUE, FALSE)
4	(o)	Attribute:	Transfer Syntax
5	(o)	Attribute:	List Of Supported Attributes
SERVICES:			
1	(m)	Ops Service	AR-Unconfirmed Send
2	(m)	Ops Service	AR-Get-Buffered-Message
3	(o)	Ops Service	AR-Status

6.1.2.2.2 System management attributes

Local AP

This attribute identifies the AP attached or configured to use the AREP using a local reference.

FAL revision

This specifies the revision level of the FAL protocol used by this endpoint. The revision level is in the AR header of all FAL-PDUs transmitted.

Dedicated

This attribute specifies whether the publisher AREP is dedicated to publish one common memory block.

Transfer syntax

This optional attribute identifies the encoding rules to be used on the AR. When not present, the default FAL Transfer Syntax of this standard is used.

List of supported attributes
This optional attribute specifies the attributes supported by the object. This list contains, at a minimum, the mandatory attributes for the class of the object.

6.1.2.2.3 Services

AR-unconfirmed send

This local service is used to send an unconfirmed service on the specified AR. The structure of the PDU is specified and described in the corresponding DL-mapping state machine.

AR-get-buffered-message

This local service is used to retrieve an APDU from the buffer used by an AR.

AR-status

This optional service is used to report status of the AR.

6.1.2.3 Application relationship ASE service specifications

6.1.2.3.1 Supported services

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

– AR-Unconfirmed Send;

– AR-Get Buffered Message;

– AR-Status.

6.1.2.3.2 AR-unconfirmed send service

6.1.2.3.2.1 Service overview

This service is used to send AR-Unconfirmed request APDUs for FAL CM ASE. The AR-Unconfirmed Send service may be requested at the publisher endpoint of a one-to-many AR.

NOTE This service is described abstractly in such a way that it is capable of operating with ARs that convey FAL APDUs through buffers. This service may be implemented in such a way that the capability is provided to load the buffer and subsequently post it for transfer by the underlying data-link layer. Alternatively, this service may be implemented such that these capabilities are combined so that the user may load the buffer and request its transfer in a single operation.

6.1.2.3.2.2 Service primitives

The service parameters for this service are shown in Table 13.

Table 13 – AR-Unconfirmed Send

Parameter name	Req	Ind
Argument		
AREP	M	M (=)
FAL APDU Body	M	M (=)

Argument

The argument contains the parameters of the service request.

FAL APDU Body

This parameter specifies the service dependent body for the APDU.

6.1.2.3.2.3 Service procedure

The AR-Unconfirmed Send Service is a service that operates through a buffer.

The requesting FAL ASE submits an AR-Unconfirmed send request primitive to its AR ASE. The AR ASE builds an AR-Unconfirmed Send request APDU.

The AR ASE replaces the previous contents of the buffer with the APDU contained in the service primitive.

The AR ASE requests the DL to transfer the data at the scheduled time. The data-link mapping indicates how the AR ASE coordinates its requests to transmit the data with the data-link layer.

NOTE The transmission schedule is managed by the underlying layer, not the AR-ASE. Refer to IEC 61158 3 11 and IEC 61158 4 11 for further details.

Upon receipt of the AR-Unconfirmed Send request APDU, the receiving AR ASE delivers an AR-Unconfirmed Send .indication primitive to the appropriate FAL ASE as indicated by the FAL Service Type Parameter.

6.1.2.3.3 AR-get buffer message service

6.1.2.3.3.1 Service overview

This local service is used by application process to request the AR ASE to retrieve a message which is being maintained in a buffer in the local data-link layer.

This service does not result in the conveyance of an APDU. It is provided so that the FAL user may access a buffer through the FAL AR.

6.1.2.3.3.2 Service primitives

The service parameters for this service are shown in Table 14.

Table 14 – AR-get buffered message service

Parameter name	Req	Cnf
Argument		
AREP	M	
Result(+)		S
Decoded Buffer Data		M
Result(-)		S
Error Info		M
NOTE The method by which a confirm primitive is correlated with its corresponding preceding request primitive is a local matter. See 1.2.		

Argument

The argument contains the parameters of the service request.

Result (+)

This selection type parameter indicates that the service request succeeded.

Decoded buffer data

This parameter specifies the user data in the FAL APDU read from the buffer.

Result (-)

This selection type parameter indicates that the service request failed.

6.1.2.3.3.3 Service procedure

This service requests the FAL to return the current contents of the buffer in a confirmation(+) primitive. If the buffer is empty, a confirmation(-) primitive is returned.

6.1.2.3.4 AR-status service

6.1.2.3.4.1 Service overview

This local service provides the FAL user notification of a status change the AREP.

6.1.2.3.4.2 Service primitives

This service parameters for this service are shown in Table 15.

Table 15 – AR-Status service

Parameter name	Ind
Argument	
AREP	M
Status code	M

Argument

This parameter carries the parameters of the service invocation.

Status code

This specifies the status change being reported. The following status codes are defined:

buffer received;

buffer update invocation;

lower layer reset.

6.1.2.3.4.3 Service procedure

This service indicates that a significant event, as defined by the status code parameter, occurred in the communication stack.

6.2 ARs

6.2.1 Buffered network-scheduled uni-directional pre-established connection (BNU-PEC) AR endpoint class specification

6.2.1.1 Class overview

This class is defined to support the “push” model for scheduled, buffered distribution of unconfirmed services to one or more application processes.

An AR ASE user wishing to broadcast new data, uses an AR ASE Service Data Unit with the specific encoding, which is specified in the IEC 61158-6-11, to its AREP for distribution. The AREP which is sending the data writes it into the data-link layer buffer, completely replacing the existing contents of the buffer. The data-link layer transfers the buffer contents at the next scheduled opportunity. When the buffer contents are transmitted, the AR ASE notifies the user of the transmission.

If the AREP which sent data receives from the AR ASE user another data before the buffer contents are transmitted, the buffer contents will be replaced with the new data and the previous data will be lost.

At the receiving endpoint, the data is received from the network, it is immediately written into the buffer, completely overwriting the existing contents of the buffer. The endpoint notifies the user that the data has arrived and delivers it to the user. If the data has not been delivered before the next data arrives, it will be overwritten by the next data and lost.

The following summarizes the characteristics of this AREP class;

Roles: Push Publisher
Push Subscriber

Cardinality: 1-to-Many

Timeliness: Optional

6.2.1.2 Formal model

FAL ASE: **AR ASE**

CLASS: Buffered Network-Scheduled Uni-directional Pre-Established AR Endpoint

CLASS ID: 64

PARENT CLASS: AR Endpoint

NETWORK MANAGEMENT ATTRIBUTES:

1	(m)	Attribute:	Role (PUSH-PUBLISHER, PUSH-SUBSCRIBER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	DL Mapping Reference

SERVICES:

1	(o)	OpsService:	Unconfirmed Send
2	(o)	OpsService:	Get Buffered Message
3	(o)	OpsService:	Status

6.2.1.3 Network management attributes

Role

This attribute specifies the role of the AREP. The valid values are:

PUSH-PUBLISHER
Endpoints of this type publish their data issuing unconfirmed service request-APDUs.

PUSH-SUBSCRIBER
Endpoints of this type receive data published in confirmed service Response-APDUs.

AREP state

This attribute specifies the state of the AREP. The values for this attribute are specified in IEC 61158-6-11.

DL mapping reference

For PUBLISHER AREPs, this attribute specifies the mapping to the transmit conveyance path. For SUBSCRIBER AREPs, this attribute specifies the mapping to the receive conveyance path. DL mapping attributes for the data-link layer (IEC 61158-3-11) are specified in IEC 61158-6-11.

6.2.1.4 Services

Unconfirmed send

This optional service is used to send an unconfirmed service on an AR. The structure of the PDU is specified and described in the corresponding DL-Mapping State Machine.

Get buffered message

This local service is used to retrieve an APDU from the buffer used by an AR.

Status

This local service provides the FAL user notification of status change the AREP.

6.3 Summary of FAL classes

This subclause contains a summary of the defined FAL Classes. The Class ID values have been assigned to be compatible with existing standards.

Table 16 provides a summary of the classes.

Table 16 – FAL class summary

FAL ASE	Class	Class ID
Common memory ASE	COMMON MEMORY	tbd
Data Type	Fixed Length & String Data Type	5
	Structure Data Type	6
	Array Data Type	12
Application Relationship	AREP	32
	BNU PEC	64

6.4 Permitted FAL services by AREP role

Table 17 below defines the valid combinations of services and AREP roles (which service APDUs and AREP with the specified role can send or receive). The Unc and Cnf columns indicate whether the service listed in the left-hand column is unconfirmed (Unc) or confirmed (Cnf).

Table 17 – Services by AREP role

	Unc	Cnf	Client		Server		Push Publ		Push Subsc		Pull Publ Mgr		Pull Publ		Pull Subsc		Report Src		Report Sink	
FAL Services			req	rcv	req	rcv	req	rcv	req	rcv	req	rcv	req	rcv	req	rcv	req	rcv	req	rcv
Common memory ASE																				
Update-Memory							X	X	X	X										
Memory-Status								X		X										
AR ASE																				
AR-Unconfirmed Send							X			X										
AR-Get Buffered Msg							X		X											
AR-Status								X		X										

Bibliography

IEC 61158-4-11, *Industrial communication networks – Fieldbus specifications - Part 4-11: Data-link layer protocol specification – Type 11 elements*

IEC 61158-6-11, *Industrial communication networks – Fieldbus specifications - Part 6-11: Application layer protocol specification – Type 11 elements*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

_____

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
P.O. Box 131
CH-1211 Geneva 20
Switzerland

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