BS EN 62769-5:2015

BSI Standards Publication

Field Device Integration (FDI)

Part 5: FDI Information Model

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 62769-5:2015. It is identical to IEC 62769-5:2015.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2015

ISBN 978 0 580 78327 2
ICS 25.040.40; 35.100

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 July 2015.

Amendments/corrigenda issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 62769-5

July 2015

ICS 25.040.40; 35.100

English Version

Field Device Integration (FDI) - Part 5: FDI Information Model (IEC 62769-5:2015)

Intégration des appareils de terrain (FDI) - Partie 5: Modèle d'Information FDI (IEC 62769-5:2015)

Feldgeräteintegration (FDI) - Teil 5: FDI-Informationsmodell (IEC 62769-5:2015)

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 62769-5:2015 E

European foreword

The text of document 65E/348/CDV, future edition 1 of IEC 62769-5, prepared by SC 65E “Devices and integration in enterprise systems” of IEC/TC 65 “Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 62769-5:2015.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2016-03-24
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2018-06-24

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 62769-5:2015 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC/TR 62541-1	NOTE	Harmonized as CLC/TR 62541-1.
IEC 62541-7	NOTE	Harmonized as EN 62541-7

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

Publication	Year	Title	EN/HD	Year
IEC 61784-1	-	Industrial communication networks - Profiles -- Part 1: Fieldbus profiles	EN 61784-1	-
IEC 61804-3	-	Function blocks (FB) for process control and EDDL - Part 3: EDDL specification and communication profiles	-	-
IEC 62541-3	-	OPC unified architecture - Part 3: Address Space Model	EN 62541-3	-
IEC 62541-4	-	OPC Unified Architecture - Part 4: Services	EN 62541-4	-
IEC 62541-5	-	OPC unified architecture - Part 5: Information Model	EN 62541-5	-
IEC 62541-6	-	OPC unified architecture - Part 6: Mappings	EN 62541-6	-
IEC 62541-8	-	OPC Unified Architecture - Part 8: Data Access	EN 62541-8	-
IEC 62541-100	-	OPC unified architecture - Part 100: Device Interface	EN 62541-100	-
IEC 62769-1	-	Field device integration (FDI) - Part 1: Overview	-	-
IEC 62769-2	-	Field Device Integration (FDI) - Part 2: FDI Client	-	-
IEC 62769-4	-	Field Device Integration (FDI) - Part 4: FDI Packages	-	-
IEC 62769-7	-	Field Device Integration (FDI) - Part 7: FDI Communication Devices	-	-

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

FIELD DEVICE INTEGRATION (FDI) –

Part 5: FDI Information Model

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 itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
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.

International Standard IEC 62769-5 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

CDV	Report on voting
65E/348/CDV	65E/425/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 the ISO/IEC Directives, Part 2.

A list of all parts in the IEC 62769 series, published under the general title *Field Device Integration (FDI)*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability 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.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning

a) Method for the Supplying and Installation of Device-Specific Functionalities, see Patent Family DE10357276;

b) Method and device for accessing a functional module of automation system, see Patent Family EP2182418;

c) Methods and apparatus to reduce memory requirements for process control system software applications, see Patent Family US2013232186;

d) Extensible Device Object Model, see Patent Family US12/893,680.

IEC takes no position concerning the evidence, validity and scope of this patent right.

The holders of these patent rights have assured the IEC that he/she is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from:

a) ABB Research Ltd
Claes Rytoft
Affolterstrasse 4
Zurich, 8050
Switzerland

b) Phoenix Contact GmbH & Co KG
Intellectual Property, Licenses & Standards
Flachsmarktstrasse 8, 32825 Blomberg
Germany

c) Fisher Controls International LLC
John Dilger, Emerson Process Management LLLP
301 S. 1st Avenue, Marshaltown, Iowa 50158
USA

d) Rockwell Automation Technologies, Inc.
1 Allen-Bradley Drive
Mayfield Heights, Ohio 44124
USA

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line data bases of patents relevant to their standards. Users are encouraged to consult the data bases for the most up to date information concerning patents.

FIELD DEVICE INTEGRATION (FDI) –

Part 5: FDI Information Model

1 Scope

This part of IEC 62769 defines the FDI Information Model. One of the main tasks of the Information Model is to reflect the topology of the automation system. Therefore it represents the devices of the automation system as well as the connecting communication networks including their properties, relationships, and the operations that can be performed on them. The types in the AddressSpace of the FDI Server constitute some kind of catalogue, which is built from FDI Packages.

The fundamental types for the FDI Information Model are well defined in OPC UA for Devices (IEC 62541-100). The FDI Information Model specifies extensions for a few special cases and otherwise explains how these types are used and how the contents are built from elements of DevicePackages.

The overall FDI architecture is illustrated in Figure 1. The architectural components that are within the scope of this document have been highlighted in this illustration.

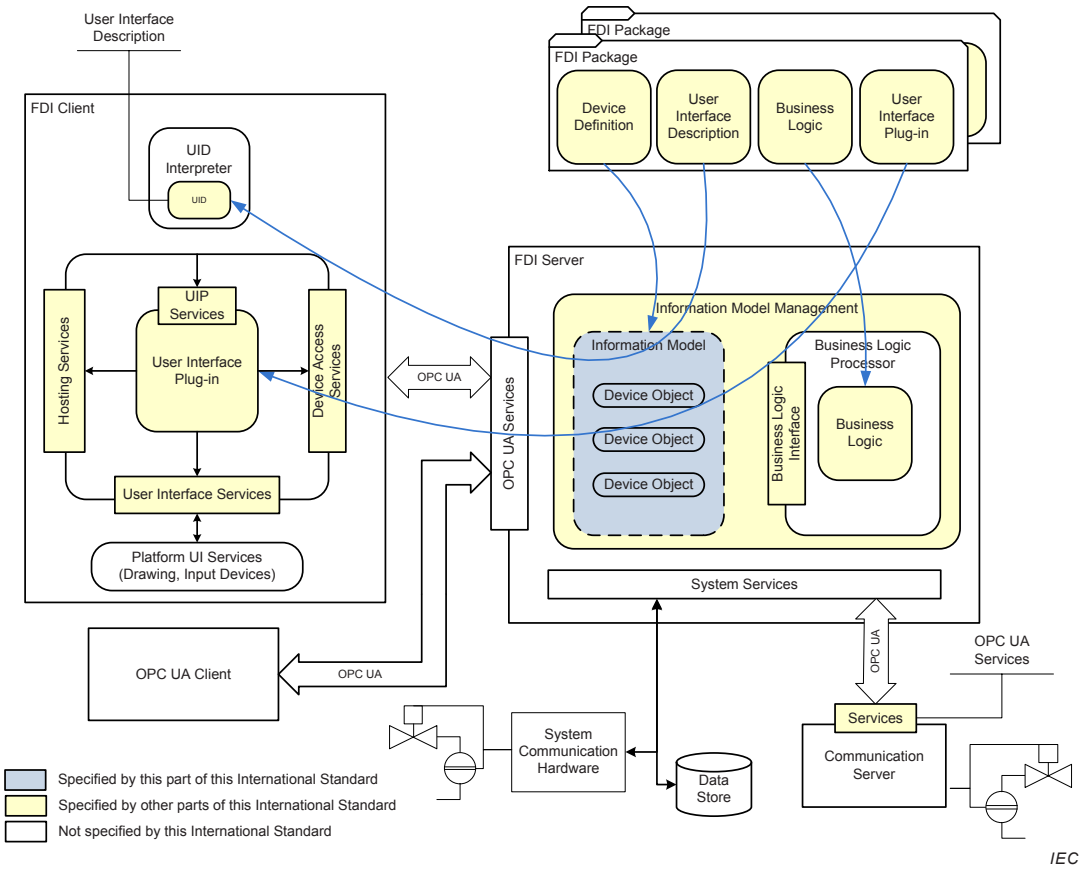

Figure 1 – FDI architecture diagram

2 Normative references

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

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61804-3[1], *Function blocks (FB) for process control and Electronic Device Description Language (EDDL) – Part 3: EDDL syntax and semantics*

IEC 62541-3, *OPC unified architecture – Part 3: Address Space* Model

IEC 62541-4, *OPC unified architecture – Part 4: Services*

IEC 62541-5, *OPC unified architecture – Part 5: Information Model*

IEC 62541-6, *OPC unified architecture – Part 6: Mappings*

IEC 62541-8, *OPC unified architecture – Part 8: Data Access*

IEC 62541-100[2], *OPC unified architecture – Part 100: OPC UA for Devices*

IEC 62769-1, *Field Device Integration (FDI) – Part 1: Overview*

NOTE IEC 62769-1 is technically identical to FDI-2021

IEC 62769-2, *Field Device Integration (FDI) – Part 2: FDI Client*

NOTE IEC 62769-2 is technically identical to FDI-2022

IEC 62769-4, *Field Device Integration (FDI) – Part 4: FDI Packages*

NOTE IEC 62769-4 is technically identical to FDI-2024

IEC 62769-7, *Field Device Integration (FDI) – Part 7: FDI Communication Devices*

NOTE IEC 62769-7 is technically identical to FDI-2027

3 Terms, definitions, abbreviated terms, acronyms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62769-1 apply.

---

1 To be published.

2 Under consideration.

3.2 Abbreviated terms and acronyms

For the purposes of this document, the abbreviated terms and acronyms given in IEC 62769-1 as well as the following apply.

HMI	Human Machine Interface
SCADA	Supervisory Control and Data Acquisition
TCP	Transmission Control Protocol

3.3 Conventions for graphical notation

OPC UA defines a graphical notation for an OPC UA AddressSpace. It defines graphical symbols for all NodeClasses and how different types of References between Nodes can be visualized. Figure 2 shows the symbols for the NodeClasses used in this standard. NodeClasses representing types always have a shadow.

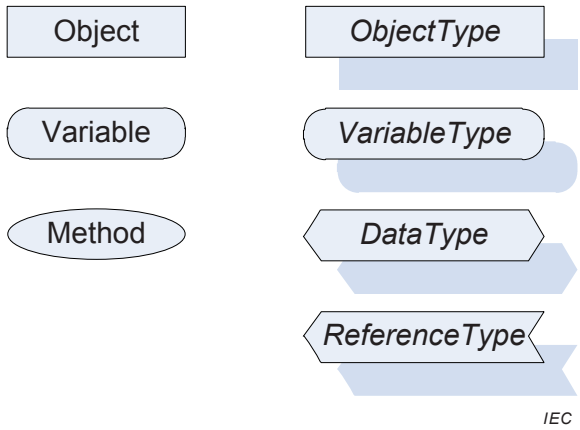

Figure 2 – OPC UA Graphical Notation for NodeClasses

Figure 3 shows the symbols for the ReferenceTypes used in this standard. The Reference symbol is normally pointing from the source Node to the target Node. The only exception is the HasSubType Reference. The most important References such as HasComponent, HasProperty, HasTypeDefinition and HasSubType have special symbols avoiding the name of the Reference. For other ReferenceTypes or derived ReferenceTypes the name of the ReferenceType is used together with the symbol.

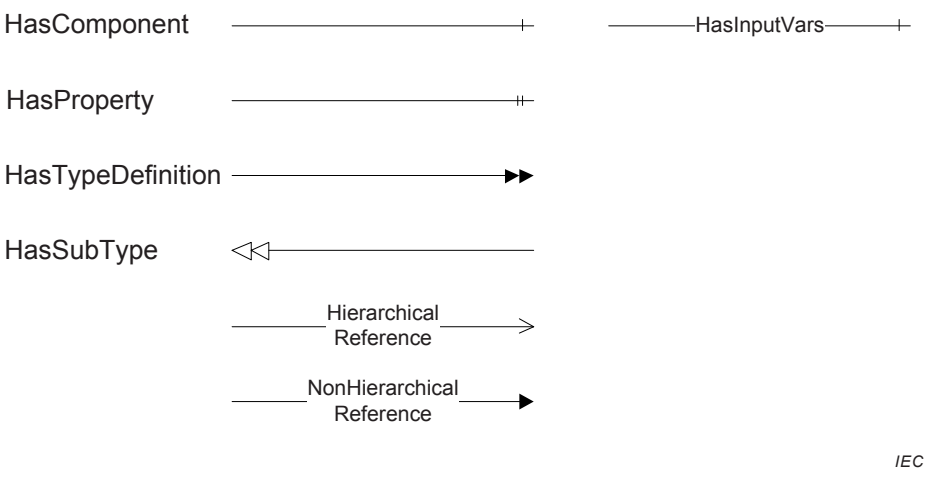

Figure 3 – OPC UA Graphical Notation for References

Figure 4 shows a typical example for the use of the graphical notation. Object_A and Object_B are instances of the ObjectType_Y indicated by the HasTypeDefinition References. The ObjectType_Y is derived from ObjectType_X indicated by the HasSubType Reference. The Object_A has the components Variable_1, Variable_2 and Method_1.

To describe the components of an Object on the ObjectType the same NodeClasses and References are used on the Object and on the ObjectType such as for ObjectType_Y in the example. The Nodes used to describe an ObjectType are instance declaration Nodes.

To provide more detailed information for a Node, a subset or all Attributes and their values can be added to a graphical symbol (see for example Variable_1, the component of Object_A in Figure 4).

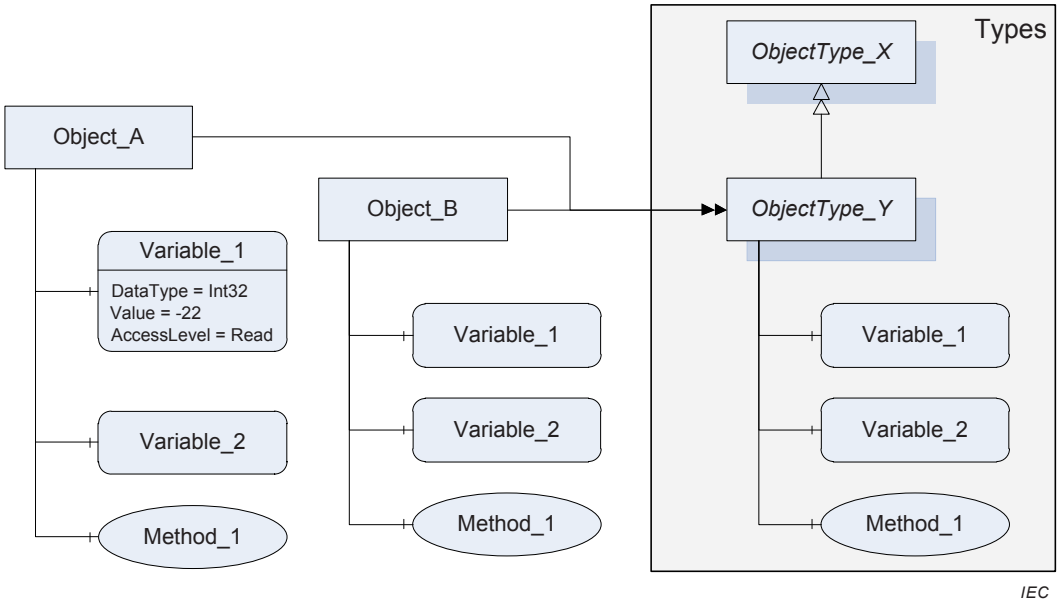

Figure 4 – OPC UA Graphical Notation Example

To improve readability, this document frequently includes the type name inside the instance box rather than displaying both boxes and a reference between them. This optimization is shown in Figure 5.

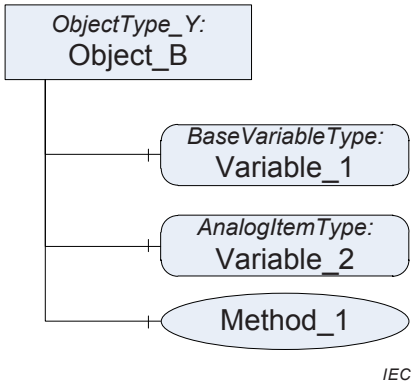

Figure 5 – Optimized Type Reference

4 Overview of OPC Unified Architecture

4.1 General

The main use case for OPC standards is the online data exchange between devices and HMI or SCADA systems. In this use case the device data is provided by an OPC server and is consumed by an OPC client integrated into the HMI or SCADA system. OPC provides functionality to browse through a hierarchical namespace containing data items and to read, write and monitor these items for data changes.

OPC UA incorporates features like Data Access, Alarms and Historical Data via platform independent communication mechanisms and generic, extensible and object-oriented modelling capabilities for the information a system wants to expose.

The current version of OPC UA defines an optimized binary TCP protocol for high performance intranet communication as well as a mapping to Web Services. The abstract service model does not depend on a specific protocol mapping and allows adding new protocols in the future. Features like security, access control and reliability are directly built into the transport mechanisms. Based on the platform independence of the protocols, OPC UA servers and clients can be directly integrated into devices and controllers.

The OPC UA information model provides a standard way for Servers to expose Objects to Clients. Objects in OPC UA terms are composed of other Objects, Variables and Methods. OPC UA also allows relationships to other Objects to be expressed.

The set of Objects and related information that an OPC UA Server makes available to Clients is referred to as its AddressSpace. The elements of the OPC UA Object Model are represented in the AddressSpace as a set of Nodes described by Attributes and interconnected by References. OPC UA defines various classes of Nodes to represent AddressSpace components most importantly Objects, Variables, Methods, ObjectTypes, DataTypes and ReferenceTypes. Each NodeClass has a defined set of Attributes.

Objects are used to represent components like folders, Devices or Networks. An Object is associated to a corresponding ObjectType that provides definitions for that Object.

Variables are used to represent values. Two categories of Variables are defined, Properties and DataVariables.

Properties are Server-defined characteristics of Objects, DataVariables and other Nodes. Properties are not allowed to have Properties defined for them. An example for Properties of Objects is the Manufacturer Property of a Device.

DataVariables represent the contents of an Object. DataVariables may have component DataVariables. This is typically used by Servers to expose individual elements of arrays and structures. This standard uses DataVariables mainly to represent the Parameters of Devices.

4.2 Overview of OPC UA Devices

The OPC Unified Architecture for Devices (DI) (IEC 62541-100) standard is an extension of the overall OPC Unified Architecture standard series and defines information models associated with Devices. IEC 62541-100 describes three models which build upon each other as follows:

- The (base) Device Model is intended to provide a unified view of devices irrespective of the underlying device protocols.
- The Device Communication Model adds Network and Connection information elements so that communication topologies can be created.

- The Device Integration Host Model finally adds additional elements and rules required for host systems to manage integration for a complete system. It allows reflecting the topology of the automation system with the devices as well as the connecting communication networks.

The Devices information model specifies different ObjectTypes and other AddressSpace elements used to represent Devices and related components such as the communication infrastructure in an OPC UA AddressSpace. The main use cases are Device configuration and diagnostic but it allows a general and standardized way for any kind of application to access Device related information.

Figure 6 shows an example for a temperature controller represented as Device Object. The component ParameterSet contains all Variables describing the Device. The component MethodSet contains all Methods provided by the Device. Both components are inherited from the TopologyElementType, which is the root Object type of the Device Object type hierarchy. Objects of the FunctionalGroupType are used to group the Parameters and Methods of the Device into logical groups. The FunctionalGroupType and the grouping concept are defined in IEC 62541-100 but the groups are DeviceType specific, i.e., the groups ProcessData and Configuration are defined by the TemperatureControllerType in this example.

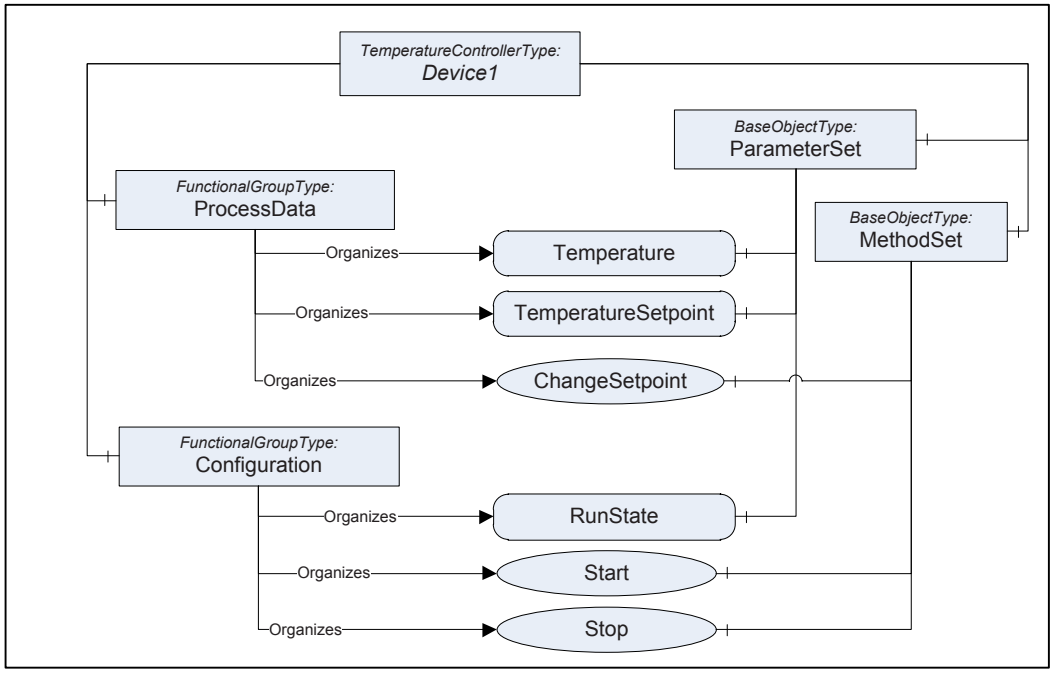

Figure 6 – OPC UA Devices Example: Functional Groups

Another IEC 62541-100 concept is illustrated in Figure 7. The ConfigurableObjectType is used to provide a way to group sub components of a Device and to indicate which types of sub components can be instantiated. The allowed types are referenced from the SupportedTypes folder. This information can be used by configuration clients to allow a user to select the type to instantiate as sub component of the Device.

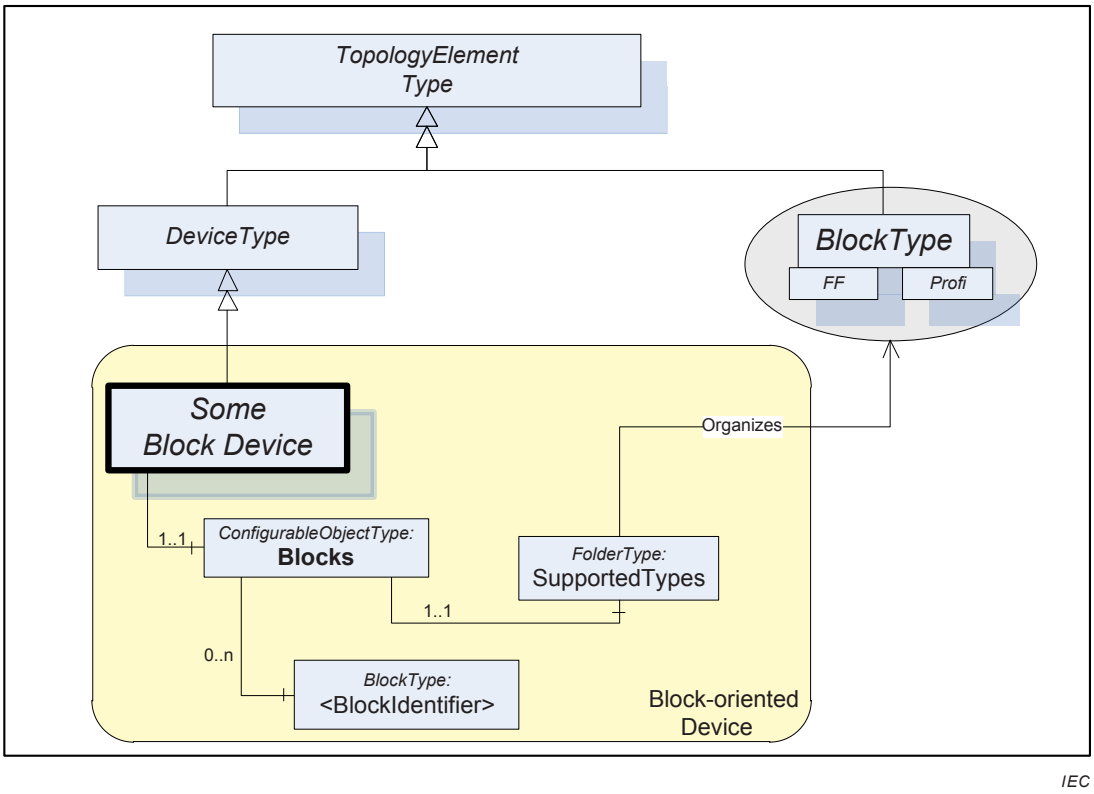

Figure 7 – OPC UA Devices example: Configurable components

The SupportedTypes folder can contain different subsets of ObjectTypes for different instances of the Block-oriented Device depending on their current configuration since the list contains only types that can be instantiated for the current configuration.

5 Concepts

5.1 General

The FDI Server provides FDI Clients access to information about Device instances and Device types regardless of where the information is stored, for example, in the Device itself or in a data store. This information is provided via OPC UA Services and is called the FDI Information Model.

The FDI Information Model specifies the entities that may be accessed in the FDI Server, including their properties, relationships, and the operations that can be performed on them. Which types of Devices or other topological elements are available in a given FDI Server is driven largely by the information in the FDI Packages.

5.2 Device topology

One of the main tasks of the Information Model is to reflect the topology of the automation system. Therefore the Information Model represents the devices of the automation system as well as the connecting communication networks. The entry point Device Topology is the starting point within the Information Model for the topology of the automation system. The entry point Communication Devices contains the communication devices that are used by the FDI Server to access the elements of the topology. Figure 8 and Figure 9 illustrate an example configuration and the configured topology as it will appear in the FDI Server AddressSpace (details left out).

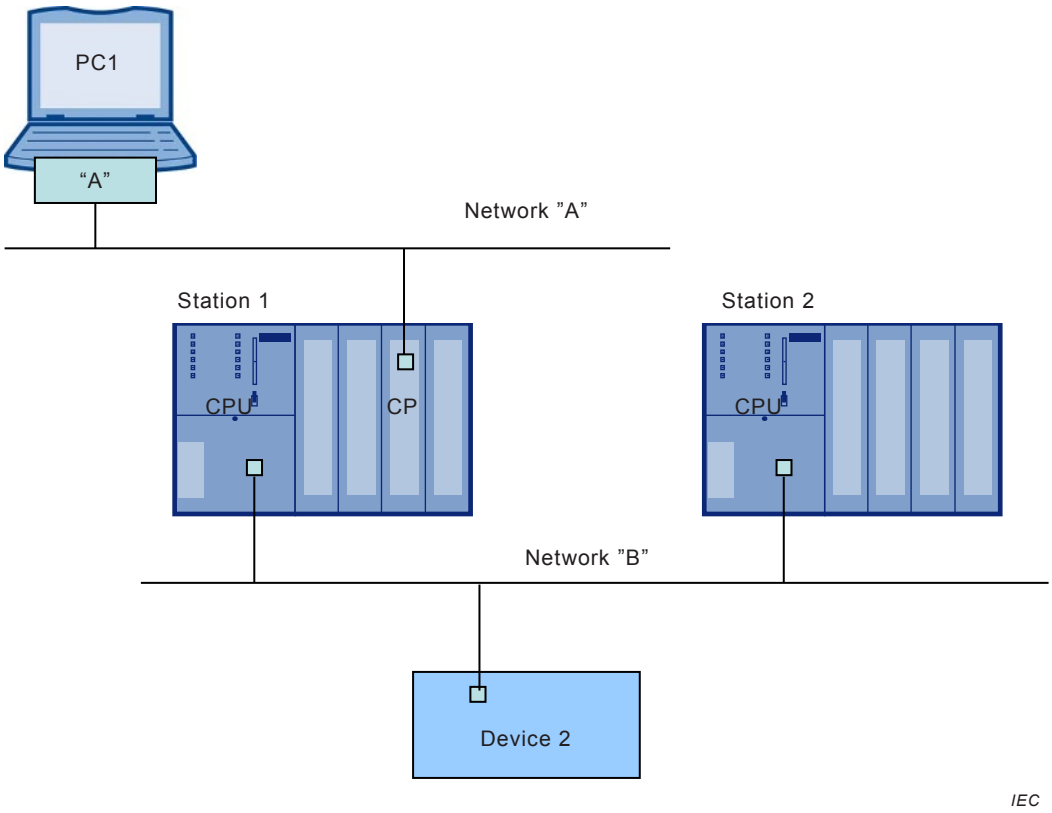

Figure 8 – Example of an automation system

The PC in Figure 8 represents the FDI Server box. The FDI Server communicates with devices connected to Network “A” via a Native Communication, and it communicates with devices connected to Network “B” via a Nested Communication.

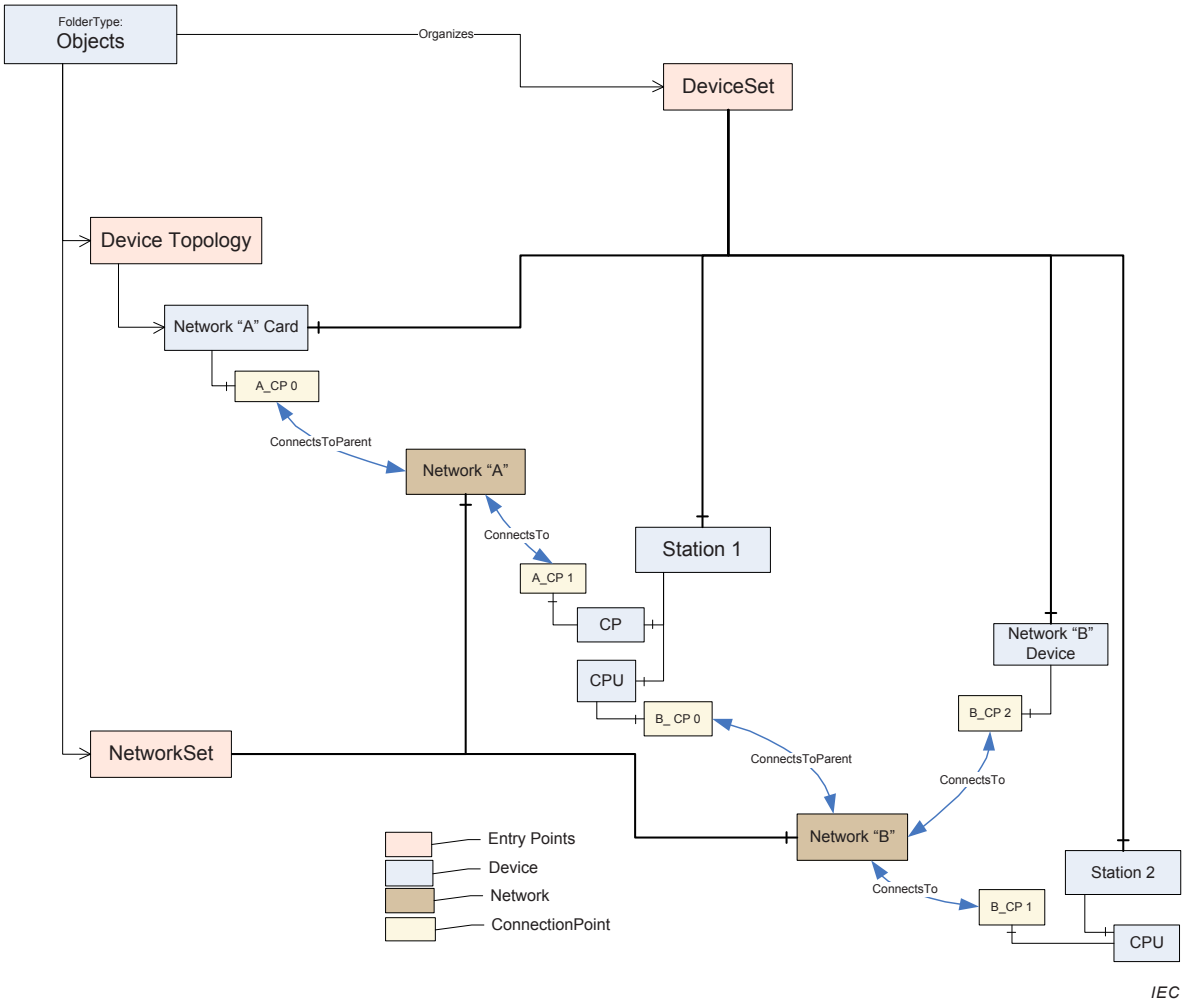

Figure 9 – Example of a Device topology

Coloured boxes are used to easily recognize the various types of information.

Brown boxes represent the networks. Light blue boxes represent the Devices and light yellow is used for Connection Points.

Light pink boxes represent the entry points that assure common behaviour across different implementations:

- DeviceTopology: Starting node for the topology configuration.
- DeviceSet: All instantiated Devices are components of this Object, i.e., they exist in the AddressSpace independently of the Device Topology.
- NetworkSet: All Networks are components of this Object.

5.3 Online/offline

Management of the Device Topology is a configuration task, i.e., the elements in the topology (Devices, Networks, and Connection Points) are usually configured "offline" and – at a later time – will be validated against their physical representative in a real network.

To support explicit access to either the online or the offline information, each element is represented by two instances that are schematically identical, i.e., there exists a ParameterSet, FunctionalGroups, and so on. A Reference connects the online and offline representation and allows navigating between them.

5.4 Catalogue (Type Definitions)

The supported (sub-types of) TopologyElements are organised as Type definitions in the OPC UA AddressSpace forming some kind of a catalogue. These definitions typically are generated based on descriptive information from FDI Packages. The Type definitions contain the Parameters, and default values for Parameters, Methods, Actions and Functional Groups including user interface elements. The FDI Server can include folders in the type model to organise the types according to manufacturer or other criteria.

Type definitions can then be used to create instances of Devices in the OPC UA AddressSpace. Instances can be created either offline or based on data determined by Scanning. Figure 10 illustrates an example of some Type definitions (details left out) as they may exist in the AddressSpace.

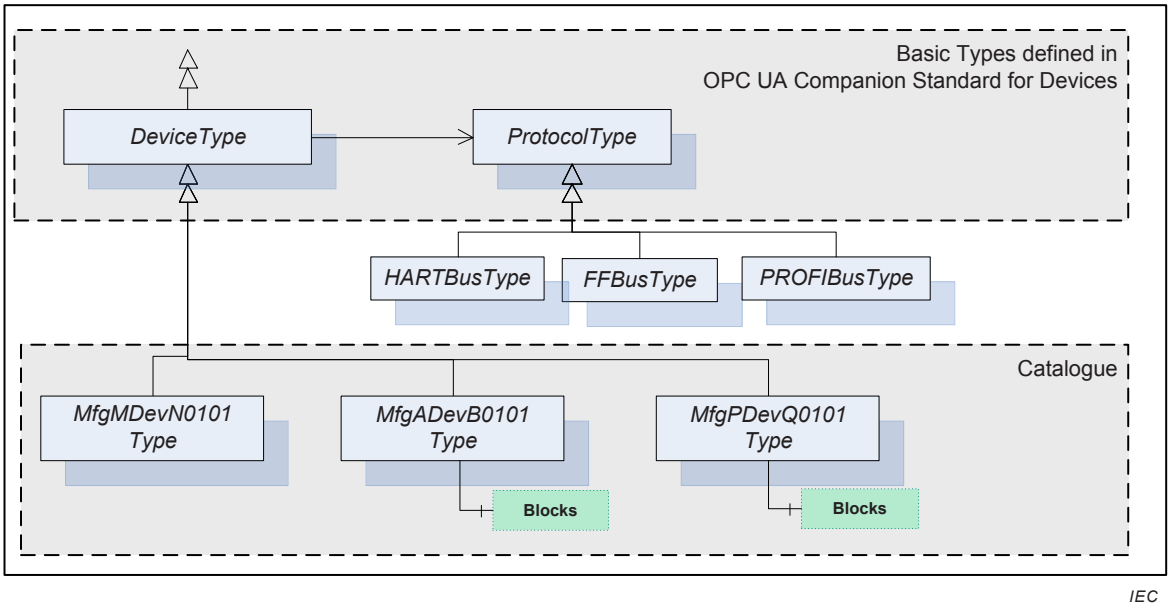

Figure 10 – Example Device Types representing a catalogue

5.5 Communication

In order to integrate Devices, the FDI Server needs to be able to communicate to them. This can be done using Native Communication or Nested Communication.

The example in Figure 9 above for instance specifies that the FDI Server has direct access to the PROFINET Network using its PROFINET network card. In order to access “Station 2”, the FDI Server has to go through Station 1, which provides the communication services for the PROFIBUS DP Network (see IEC 61784-1, CPF 3). This can be achieved through Nested Communication, which is specified in IEC 62769-4. Communication Devices and Communication Servers are specified in IEC 62769-7.

6 AddressSpace organization

To promote interoperability of FDI Clients and FDI Servers, a set of Objects and relationships are defined in the following subclauses. FDI Servers can implement a subset of these standard Nodes, depending on their capabilities.

Based on OPC UA rules, an OPC UA Server separates the AddressSpace in two parts:

a) The Types-part contains information about all components that have been generated based on descriptive information from FDI Packages (see 5.4).

b) The Objects-part contains the Device Topology with all instantiated components. All instances of the AddressSpace are related to a type of the Types folder

   – The entry points DeviceSet, NetworkSet, and DeviceTopology are formally defined in IEC 62541-100.

      • DeviceTopology is used to aggregate the top level Networks that provide access to all instances that constitute the Device Topology ((sub-)networks, devices and communication elements). Some example elements are shown here and highlighted using the green colour.

      • All instantiated Devices are components of the DeviceSet Object, i.e., they exist in the AddressSpace independent of the Device Topology. All Networks are components of the NetworkSet Object.

FDI Servers can either automatically create Device Objects or they may only show the available types (SupportedTypes folder) and leave it to the user to create proper instances. When using native communication the system will typically provide the Device Topology without having to have it configured by the FDI Client.

7 Device Model for FDI

7.1 General

As mentioned above, IEC 62541-100 specifies the fundamental types needed for FDI like the TopologyElementType, the DeviceType and the ProtocolType (the fieldbus protocol). Clause 7 briefly repeats important design elements specified in IEC 62541-100 and specifies additional types that are not in IEC 62541-100.

7.2 Online/offline

All elements that appear in the Device Topology (Devices, Networks, and Connection Points) including their relationship correspond to information stored in the FDI Server's configuration database. Management of these elements most commonly requires access to the physical component/device (called online data in this standard) and also the storage and administration of related data in a configuration database (called offline data).

To support explicit access to either the online or the offline information, each element is represented by two instances that are schematically identical, i.e., there exists a ParameterSet, FunctionalGroups, and so on. A Reference connects online and offline representation and allows to navigate between them. This is illustrated in Figure 11.

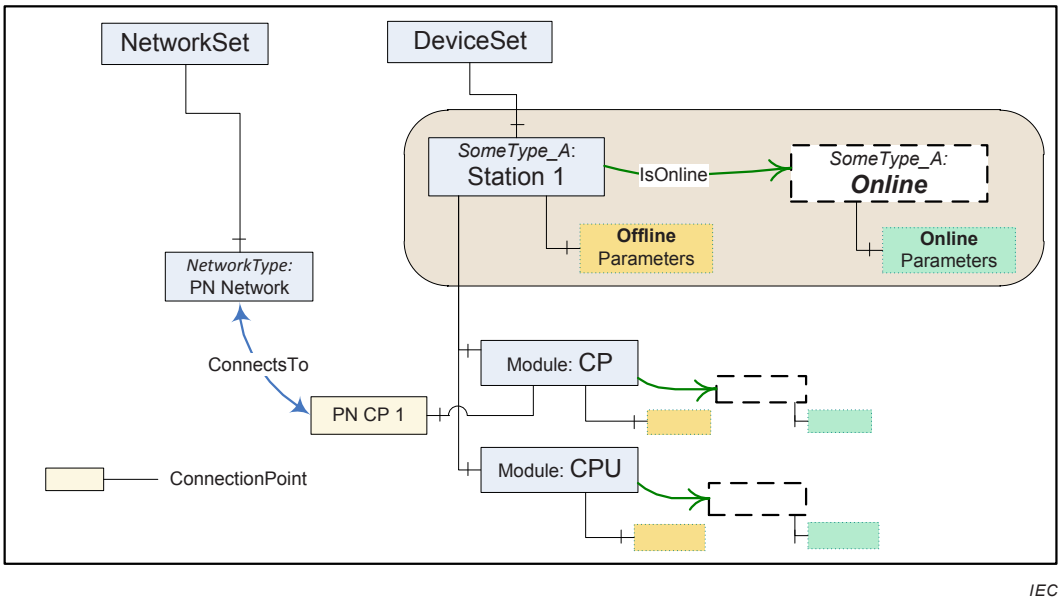

Figure 11 – Online component for access to device data

Support of online/offline is mandatory for FDI Servers. Detailed information of the model and the formal definitions are specified in IEC 62541-100.

7.3 Device health

The DeviceHealth Property indicates the status of a device as defined by NAMUR NE107. FDI Clients can read or monitor this Property to determine the device condition.

Servers determine the health status using the EDD METHOD GetHealthStatus defined in IEC 62769-4. The frequency at which Servers actually examine the health status may vary from several seconds up to minutes.

Support of the DeviceHealth Property is mandatory for Device Objects. Detailed information of the model and the formal definition are in IEC 62541-100.

7.4 User interface elements

7.4.1 General

IEC 62541-100 defines in an abstract way, how Servers can expose user interface elements for Clients to display a user interface specific to a FunctionalGroup of a TopologyElement.

Subclause 7.4 specifies two concrete user element types: descriptive user interface elements (UIDs) and programmed (executable) user interface elements (UIPs). UIPs are never referenced directly from a FunctionalGroup. They are always indirectly referenced from a UID by means of their UipId.

Figure 12 illustrates the type hierarchy of the user interface elements defined in IEC 62541-100 and in this standard.

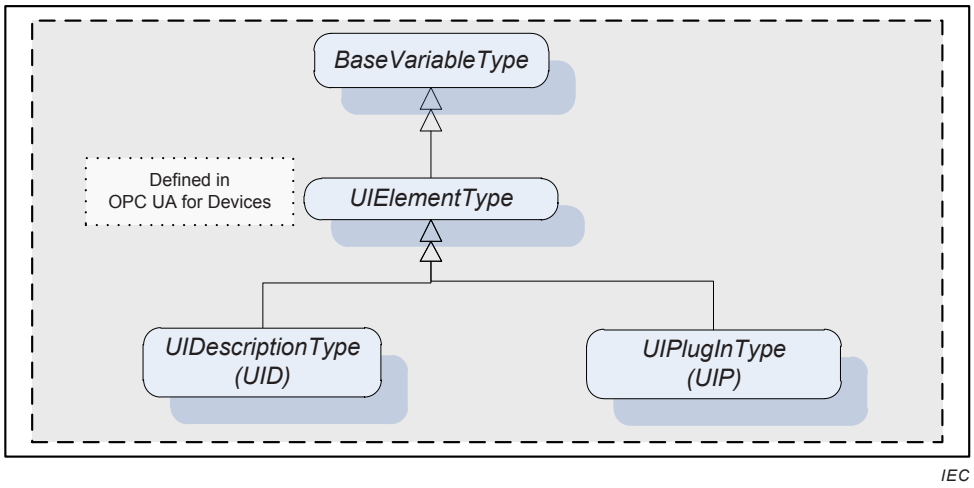

Figure 12 – Hierarchy of user interface Types

7.4.2 UI Description Type

FDI Servers may provide a descriptive user interface element (a UID) for each FunctionalGroup. Such an element will be rendered by the FDI Client. The UIDescriptionType is formally specified in Table 1.

Table 1 – UIDescriptionType Definition

Attribute	**Value**				
BrowseName	UIDescriptionType				
DataType	String				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherits the Properties of the UIElementType defined in IEC 62541-100.					

The Value Attribute provides the UID as a String containing an XML Element. See IEC 62769-2 for the syntax of UID elements. The XML Schema for the UID Value of all exposed devices adheres to the same FDI Technology Version as indicated by the FDIServerVersion Property (see Clause 14).

7.4.3 UI Plug-in Type

A User Interface Plug-in (UIP) is a software module that is hosted and run by an FDI Client. In contrast to a User Interface Description (UID) it is an executable UI element.

Details on hosting and running Plug-ins are specified in IEC 62769-2. The UIPlugInType is formally specified in Table 2.

Table 2 – UIPlugInType Definition

Attribute	Value				
BrowseName	UIPlugInType				
DataType	Byte				
ValueRank	1 – one dimensional array				
ArrayDimensions	Uint32[1] – the length (number of bytes) of the array				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherits the Properties of the UIElementType defined in IEC 62541-100.					
HasProperty	Variable	UIPVariantVersion	String	PropertyType	Mandatory
HasProperty	Variable	FDITechnologyVersion	String	PropertyType	Mandatory
HasProperty	Variable	RuntimeId	String	PropertyType	Mandatory
HasProperty	Variable	CpuInformation	String	PropertyType	Mandatory
HasProperty	Variable	PlatformId	String	PropertyType	Mandatory
HasProperty	Variable	Style	String	PropertyType	Mandatory
HasProperty	Variable	StartElementName	String	PropertyType	Mandatory
HasComponent	Object	Documentation		FolderType	Optional

UIPs may exist in multiple variants for different platforms or supporting different versions. The UipId (a unique identifier defined in the FDI package) identifies the UIP, not a specific variant.

UIPs do not have to be exposed in the AddressSpace. With the UipId the FDI Client can retrieve the NodeIds of UIP Variants and their Properties by calling the OPC UA TranslateBrowsePathToNodeIds Service with the Device NodeId as the startingNode and the following list of relative names:

- "UIPSet/<UipId>"
- "UIPSet/<UipId >/RuntimeId"
- "UIPSet/<UipId >/CpuInformation"
- "UIPSet/<UipId >/PlatformId"
- "UIPSet/<UipId >/FDITechnologyVersion"
- "UIPSet/<UipId >/Style"
- "UIPSet/<UipId >/StartElementName"
- "UIPSet/<UipId >/UIPVariantVersion"

NOTE "UIPSet" is an identifier for the Server and does not have to be a Node in the AddressSpace.

The FDI Server returns arrays of NodeIds for each relative name. The number of entries in each array matches the number of UIP Variants for the UipId. The FDI Client can read the property values using the received NodeIds and choose the appropriate UIP Variant based on FDITechnologyVersion, RuntimeId, CpuInformation, and PlatformId.

The Value Attribute provides the UIP executable. The exact representation is technology dependent (see IEC 62769-6). The ArrayDimensions Attribute shall specify the size (number of bytes) of the UIP.

FDI Clients need to be able to handle large UIPs. Reading large UIPs with a single Read operation may not be possible due to configured limits in either the FDI Client or the FDI Server stack. The default maximum size for an array of bytes is 1 MegaByte. FDI Clients can use the IndexRange in the OPC UA Read Service (see IEC 62541-4) to read a UIP in – for instance – one megabyte chunks. It is up to the FDI Client whether it starts without index and repeats with an indexRange only after an error or whether it always uses an indexRange.

The following Properties help the FDI Client to identify which UIP fits best to its environment:

- UIPVariantVersion:
  The version of this UIP Variant.
- FDITechnologyVersion:
  FDI Technology Version according to which the UIP is developed. A UIP shall always be capable of running in a client/server system with the same major version and different minor/maintenance version.
- RuntimeId:
  Runtime environment of the UIP as specified in IEC 62769-6.
- CpuInformation:
  Provides additional information about the execution environment associated with the RuntimeId. The allowed values are specified in IEC 62769-6.
- PlatformId defines the type of platform on which this UIP Variant is supported. An FDI Client can choose a particular UIP Variant if it matches the FDI Client's platform (see IEC 62769-4 for the concrete definitions).
  - "Workstation"– with regular screen resolution capabilities, memory capabilities, input devices available (like mouse and keyboard)
  - "Mobile"– limited screen resolution, memory and input devices possible
- Style:
  Defines whether the UIP shall be run "modal" or "modeless" as defined in IEC 62541-4. Currently, the values "Dialog" and "Window" are defined. While "Dialog" requires a modal window, a UIP with style "Window" will be invoked either in a modal or a non-modal window as defined in IEC 62769-2.
- StartElementName:
  Element needed to start this UIP Variant. IEC 62769-6 specifies how this information is used when activating the UIP.

Documents provided for a UIP Variant are exposed as Variables organized in the Documentation folder. In most cases they will represent a product manual, which can exist as a set of individual documents. The information can be retrieved by reading the Variable value which is represented as a ByteString. The complete ByteString shall be interpreted as a PDF file. FDI Clients need to be aware that the contents that these variables represent may be large. Reading large values with a single Read operation may not be possible due to configured limits in either the FDI Client or the FDI Server stack. The default maximum size for an array of bytes is 1 MegaByte. It is recommended that FDI Clients use the IndexRange in the OPC UA Read Service (see IEC 62541-4) to read these Variables in chunks, for example, one megabyte chunks.

7.5 Type-specific support information

Each DeviceType may have a set of additional data. These are mainly images, documents, or protocol-specific data. The various types of information are organized into different folders.

See IEC 62541-100 for the formal definition of support information.

7.6 Actions

7.6.1 Overview

Actions are operations that are executed in the FDI Server on behalf of a topology element. Once invoked with the InvokeAction Method, Actions can make various state transitions until completed. The actual state is accessible via a transient, non-browsable Variable the NodeId of which is returned by the InvokeAction Method.

FDI Clients can subscribe to this Variable to receive updates concerning the Action execution (Action data). Action data may report a state transition or a request to the FDI Client that input is necessary for continuation. The FDI Client can resume the execution with the RespondAction Method and submit the requested data.

Figure 13 illustrates how Actions are integrated into a TopologyElement. It is formally defined in Table 3.

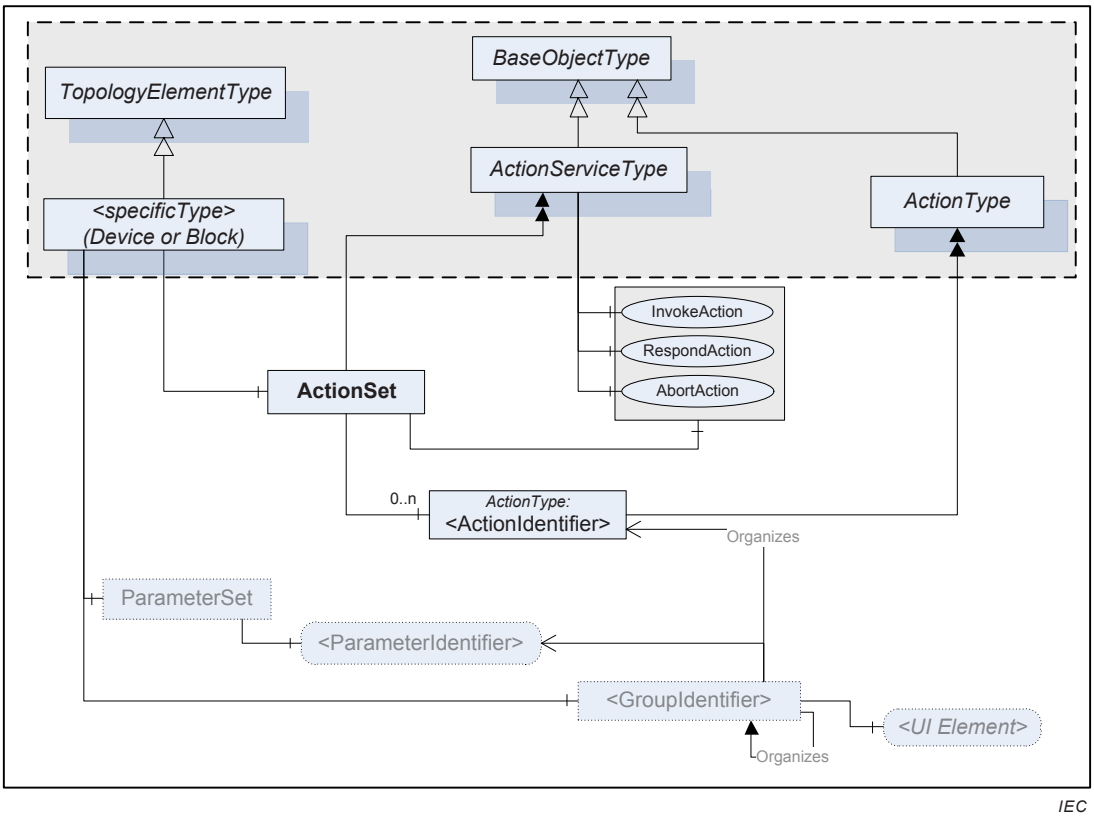

Figure 13 – Integration of Actions within a TopologyElement

Table 3 – TopologyElementType with additions for Actions

Attribute	**Value**				
BrowseName	TopologyElementType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherits the Properties of the BaseObjectType defined in [IEC 62541-5]					
HasComponent	Object	ParameterSet		BaseObjectType	Optional
HasComponent	Object	MethodSet		BaseObjectType	Optional
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OptionalPlaceHolder
HasComponent	Object	Identification		FunctionalGroupType	Optional
HasComponent	Object	Lock		LockingServicesType	Optional
HasComponent	Object	ActionSet		ActionServiceType	Optional

ActionSet is used to aggregate the Actions for a specific TopologyElement. It is added to the TopologyElementType so that it can be used on any sub-type as well. This Object is only available in an instance if Actions exist for this TopologyElement.

Action related extensions to the FunctionalGroup ObjectType are formally defined in Table 4.

Table 4 – FunctionalGroupType with additions for Actions

Attribute	Value				
BrowseName	FunctionalGroupType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Inherits the Properties of the FolderType defined in [IEC 62541-5]					
HasComponent	Object	<GroupIdentifier>		FunctionalGroupType	OptionalPlaceHolder
Organizes	Variable	<ParameterIdentifier>		BaseDataVariableType	OptionalPlaceHolder
Organizes	Method	<MethodIdentifier>			OptionalPlaceHolder
HasComponent	Variable	UIElement	BaseDataType	UIElementType	Optional
Organizes	Object	<ActionIdentifier>		ActionType	Optional

<ActionIdentifier> refers to Actions available and exposed for this FunctionalGroup.

7.6.2 Action Type

This ObjectType defines the structure of an Action. It is formally defined in Table 5.

Table 5 – ActionType Definition

Attribute	Value				
BrowseName	ActionType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType is defined in IEC 62541-5.					

The FDI Client determines the required invocation arguments for an Action from the UID.

7.6.3 ActionService Type

The ActionServiceType defines the Methods to invoke and control Actions. Instances of this type aggregate the Actions for a specific topology element. The ActionServiceType is formally defined in Table 6. Its use is illustrated in Figure 13.

Table 6 – ActionServiceType Definition

Attribute	Value				
BrowseName	ActionServiceType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in IEC 62541-5.					
HasComponent	Method	InvokeAction			Mandatory
HasComponent	Method	RespondAction			Mandatory
HasComponent	Method	AbortAction			Mandatory
HasComponent	Object	<ActionIdentifier>		ActionType	Optional

The ActionServiceType and each instance of this Type share the same Methods. The NodeId of these Methods will be fixed and defined in this standard. FDI Clients therefore do not have to browse for these Methods. They can use the fixed NodeId as the MethodId of the Call Service.

The OPC UA StatusCode Bad_MethodInvalid shall be returned from the Call Service for elements where the ActionService Methods are not supported.

<ActionIdentifier> stands for one or several Actions. Actions (Objects of ActionType) exist only in instances of the ActionServiceType, i.e., when an instance of this ObjectType is added to a TopologyElement. No Actions (Objects of ActionType) exist in the ActionServiceType itself.

7.6.4 ActionService Object

The support of the ActionService for an Object is declared by aggregating an instance of the ActionService Type as illustrated in Figure 14.

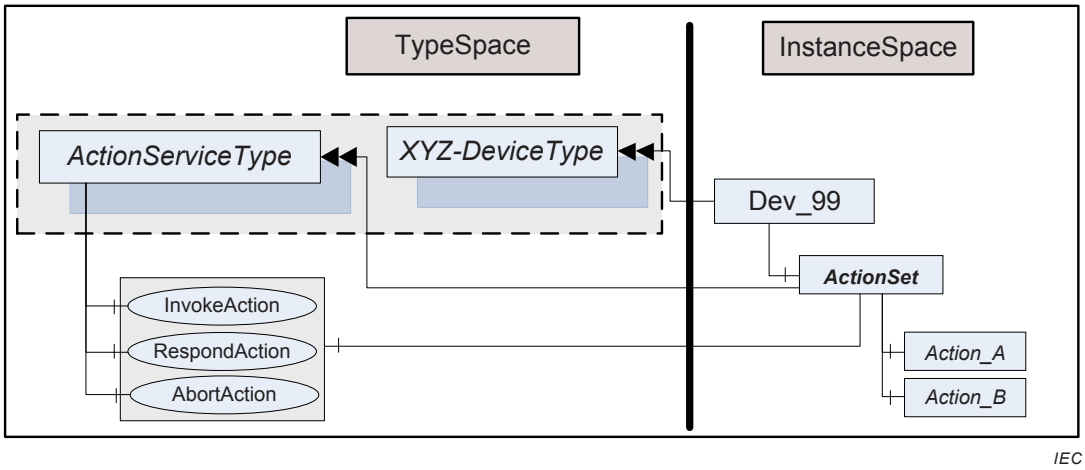

Figure 14 – Action Service

This Object is used as container for the ActionService Methods and shall have the BrowseName ActionSet. It is formally defined in Table 28. HasComponent is used to reference from a TopologyElement (for example, a Device) to its “ActionService” Object.

The ActionServiceType and each ActionSet Object share the same Methods. Actions will typically be shared by all instances of the same Device Type.

7.6.5 InvokeAction Method

InvokeAction is used to start an Action. It immediately returns after the state machine has been created. The “ActionSet” component of the TopologyElement (Device) on behalf of which the Action shall be invoked is specified via the ObjectId argument of the Call Service.

Explicit locking is required. If the Device has not been locked, the FDI Server will reject the request.

After InvokeAction returns, the FDI Client shall subscribe to the Value Attribute of the ActionNodeId. The Value Attribute contains an XML element (DataType = String)) that reflects the current state of the Action as well as additional data (depending on state). The FDI Client therefore will get a DataChange notification whenever the state of the Action changes.

See IEC 62769-2 for the Action state diagrams as well as the XML Schema of the ActionNodeId value.

FDI Servers shall cache Action state data for an appropriate period of time (a few seconds) so that no state information is lost until the FDI Client had a chance to subscribe.

The signature of this Method is specified below. Table 7 and Table 8 specify the arguments and AddressSpace representation, respectively.

Signature

```
InvokeAction(
  [in]  String       ActionName,
  [in]  String       MethodArguments,
```

```
    [out] NodeId        ActionNodeId,
    [out] Int32         InvokeActionError);
```

Table 7 – InvokeAction Method Arguments

Argument	Description
ActionName	String portion of the BrowseName of the Action as used in the ActionServiceType instance.
MethodArguments	XML document that contains the Action input arguments (if any). The ListOfActionArguments XML Schema defined in IEC 62769-2 is used for the MethodArguments parameter.
ActionNodeId	Non-browsable node of type Variable. This node is used both to identify the instance of the Action state machine and for access to the Action state information.
InvokeActionError	0 – OK. -1 - E_LockRequired – the element is not locked as required -2 - E_UnknownAction – the passed name is not a valid action for this element.

Table 8 – InvokeAction Method AddressSpace Definition

Attribute	Value				
BrowseName	InvokeAction				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.6 RespondAction Method

RespondAction is used by the FDI Client to provide the requested value or selection of a UI function request (if applicable). The “ActionSet” component of the TopologyElement on behalf of which the Action had been invoked is specified via the ObjectId argument of the Call Service.

The signature of this Method is specified below. Table 9 and Table 10 specify the arguments and AddressSpace representation, respectively.

Signature

```
RespondAction(
    [in]  NodeId        ActionNodeId,
    [in]  String        Response,
    [out] Int32         RespondActionError);
```

Table 9 – RespondAction Method Arguments

Argument	Description
ActionNodeId	NodeId of a transient Variable that represents and identifies the executing Action. This Id is returned by the InvokeAction Method.
Response	XML document that contains the response (value or selection). See IEC 62769-2 for the Xml Schema for the Response parameter.
RespondActionError	0 – OK -1 – E_InvalidAction – the NodeId does not refer to an existing action -2 – E_InvalidResponse – the passed response data could not be interpreted

Table 10 – RespondAction Method AddressSpace Definition

Attribute	Value				
BrowseName	RespondAction				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

7.6.7 AbortAction Method

AbortAction is used by the FDI Client to abort an Action execution. This Method call immediately returns. The FDI Client is informed via a notification event on the ActionNodeId Variable when the Action enters the Aborting state.

The “ActionSet” component of the TopologyElement on behalf of which the Action had been invoked is specified via the ObjectId argument of the Call Service.

The signature of this Method is specified below. Table 11 and Table 12 specify the arguments and AddressSpace representation, respectively.

Signature

```
AbortAction(
  [in]  NodeId       ActionNodeId,
  [out] Int32        AbortActionError);
```

Table 11 – AbortAction Method Arguments

Argument	Description
ActionNodeId	NodeId of a transient Variable that represents and identifies the executing Action. This Id is returned by the InvokeAction Method.
AbortActionError	0 – OK -1 – E_InvalidAction – the NodeId does not refer to an existing action

Table 12 – AbortAction Method AddressSpace Definition

Attribute	Value				
BrowseName	AbortAction				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

8 Network and connectivity

Network and connection information elements are required to create communication topologies.

A Network represents the communication means for Devices that are connected to it. It includes wired and wireless technologies. ConnectionPoints represent the interface (interface card) of a Device to a Network. A specific sub-type shall be defined for each protocol.

These elements are described and formally defined in IEC 62541-100.

9 Utility functions

9.1 Overview

Clause 9 provides specific services for certain FDI information elements.

9.2 Locking

Locking is the means to avoid concurrent modifications to a Device or Network and their components. FDI Clients shall use the Locking Services for any changes (e.g., write operations and Action invocations).

The main purpose of locking a Device is avoiding concurrent device modifications. The main purpose of locking a Network is avoiding concurrent topology changes.

When locking a Device, the lock always applies to both the online and the offline version.

When locking a Modular Device, the lock applies to the complete device (including all modules). Equally, when locking a Block Device, the lock applies to the complete device (including all blocks).

If no lock is applied to the top-level Device (for Modular Device or for Block Device), the sub-devices or blocks, respectively, can be locked independently.

When locking a Network, the lock applies to the Network and all connected Devices. If any of the connected Devices provides access to a sub-ordinate Network (such as a Gateway), the sub-ordinate Network and its connected Devices are locked as well.

The LockingService is fully described and formally defined in IEC 62541-100.

9.3 EditContext

9.3.1 Overview

An EditContext can be used to make changes to Variable values visible to the Server without applying them to the Device. The FDI Server provides the EditContext concept to support Clients in their editing task.

The EditContext is specified in IEC 62541-3. Following is the OPC UA Information Model including the Methods to maintain EditContext instances.

EditContext is exposed as an AddIn capability which is comparable to the interface technology found in some programming languages. The EditContext service is modelled as an ObjectType and instances of this type are added to the Device with a pre-defined BrowseName. Additional AddIn examples are defined in IEC 62541-100.

When reading or subscribing to Variable values registered in an EditContext, the following FDI-specific StatusCodes may occur (see Clause 11):

- Good_Edited distinguishes values that have been edited but have not been written to the Device.
- Uncertain_DominantValueChanged indicates that dependent values are invalid and will be recalculated after the dominant value has been applied to the device. In the offline case the dependent value has to be written by the Client as well.
- Good_DependentValueChanged indicates that a dependent value has been changed but the change has not been applied to the device.

9.3.2 EditContext Type

The EditContextType comprises the EditContext Methods. It is formally defined in Table 13.

Table 13 – EditContextType Definition

Attribute	Value				
BrowseName	EditContextType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in IEC 62541-5.					
HasComponent	Method	GetEditContext			Mandatory
HasComponent	Method	RegisterNodesById			Mandatory
HasComponent	Method	RegisterNodesByRelativePath			Mandatory
HasComponent	Method	Apply			Mandatory
HasComponent	Method	Reset			Mandatory
HasComponent	Method	Discard			Mandatory

The StatusCode Bad_MethodInvalid shall be returned from the Call Service for elements where the EditContext Methods are not supported. Bad_UserAccessDenied shall be returned if the Client User does not have the permission to call the Methods.

9.3.3 EditContext Object

The support of EditContext for an Object is declared by aggregating an instance of the EditContextType as illustrated in Figure 15.

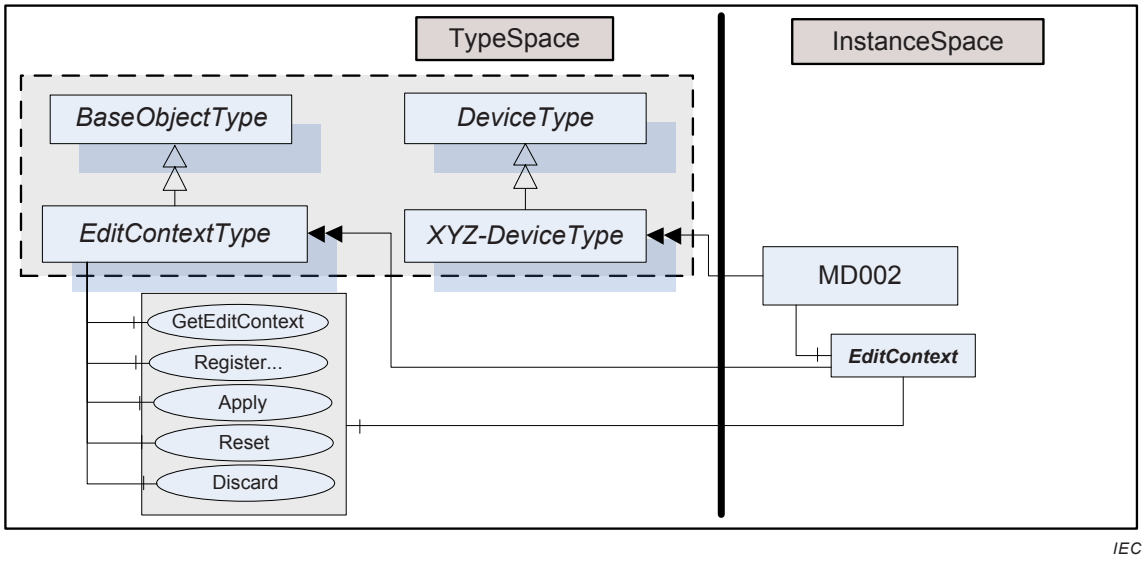

Figure 15 – EditContext type and instance

This Object is used as container for the EditContext Methods and shall have the BrowseName EditContext. HasComponent is used to reference from a Device to its “EditContext” Object.

The EditContextType and each instance may share the same Methods.

9.3.4 GetEditContext Method

Returns an EditContext as specified in IEC 62541-3.

The signature of this Method is specified below. Table 14 and Table 15 specify the arguments and AddressSpace representation, respectively.

Signature

```
GetEditContext(
  [in]  String       ParentId,
  [in]  WindowMode   TargetWindowMode,
  [out] String       EditContextId,
  [out] Int32        GetEditContextStatus);
```

Table 14 – GetEditContext Method Arguments

Argument	Description
ParentId	If Null, a root instance is requested. Otherwise, the Client passes the identifier of a previously acquired EditContext, indicating that it will create a sub-window.
TargetWindowMode	An enumeration that indicates the User Interface element used for this context. 1 – Modal Window 2 – NonModal Window 3 – UIP
EditContextId	A string identifier created by the Server.
GetEditContextStatus	0 – OK -1 – E_NotSupported – the element does not support the EditContext Service

Table 15 – GetEditContext Method AddressSpace Definition

Attribute	Value				
BrowseName	GetEditContext				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.3.5 RegisterNodes Method

This Method is used to register Nodes with an EditContext. It returns new NodeIds that have to be used to address this Node within the EditContext.

The signature of this Method is specified below. Table 16 and Table 17 specify the arguments and AddressSpace representation, respectively.

Signatures

```
RegisterNodes(
  [in]  String                    EditContextId,
  [in]  RegistrationParameters[]  NodesToRegister,
  [out] RegisterNodesResult       RegisterNodesStatus);
```

Table 16 – RegisterNodes Method Arguments

Argument	Description
EditContextId	Identifier of an EditContext that was previously acquired with a GetEditContext call.
NodesToRegister	An array of structures for each node to register.
RegisterNodesStatus	A structure with overall execution status and result data for each Node to register.

Table 17 – RegisterNodes Method AddressSpace Definition

Attribute	Value				
BrowseName	RegisterNodes				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

The RegistrationParameters DataType defines a single Node to be registered. Its elements are defined in Table 19.

Table 18 – RegistrationParameters DataType Structure

Name	Type	Description
RegistrationParameters	structure	This structure specifies one of the nodes to register.
path	RelativePath	RelativePath for this Node to register. The RelativePath type is defined in IEC 62541-4.
selectionFlags	UInt32	A bit mask that identifies the EditContext-specific NodeIds to be returned in the RegisterNodesResult structure. The value of this parameter shall contain at least one of the following values. No value will be rejected with E_InvalidSelectionFlags. Bit Value — NodeId to return 0x0000 0001 — Online / ContextNodeId 0x0000 0002 — Online / DeviceNodeId 0x0000 0004 — Offline / ContextNodeId 0x0000 0008 — Offline / DeviceNodeId

The RegisterNodesResult DataType includes overall status information and result data for each Node to be registered. Its elements are defined in Table 19.

Table 19 – RegisterNodesResult DataType Structure

Name	Type	Description
RegisterNodesResult	structure	This structure specifies the registration result.
status	Int32	0 – OK – the field registeredNodes contains a result for each Node to register -1 – E_InvalidId – the specified EditContext is unknown the registeredNodes field is empty
registeredNodes	RegisteredNode[]	The list contains EditContext-specific NodeIds for the registered Node. The Client has to use these NodeIds for all subsequent OPC UA Service calls
nodeStatus	Int32	0 – OK -1 – E_InvalidNode – an invalid Node has been registered. See IEC 62541-3 for details. -2 – E_InvalidSelectionFlags – the RegistrationParameters for this Node contained no value
onlineContextNodeId	NodeId	This NodeId shall be used to address the online representation of the Node in the EditContext. See IEC 62541-3 for details.
onlineDeviceNodeId	NodeId	This NodeId shall be used to address the online representation of the Node in the Device. See IEC 62541-3 for details.
offlineContextNodeId	NodeId	This NodeId shall be used to address the online representation of the Node in the EditContext. See IEC 62541-3 for details.
offlineDeviceNodeId	NodeId	This NodeId shall be used to address the online representation of the Node in the Device. See IEC 62541-3 for details.

9.3.6 Apply Method

This Method is used to apply values that have been modified in the EditContext.

The signature of this Method is specified below. Table 20 and Table 21 specify the arguments and AddressSpace representation, respectively.

Signature

```
Apply(
  [in]  String        EditContextId,
  [out] ApplyResult   ApplyStatus);
```

Table 20 – Apply Method Arguments

Argument	Description
EditContextId	Identifier of an EditContext that was previously acquired with a GetEditContext call.
ApplyStatus	A structure with overall execution status and status information for each transferred Variable.

Table 21 – Apply Method AddressSpace Definition

Attribute	Value				
BrowseName	Apply				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

The ApplyResult DataType includes overall Apply status information and status information for each Variable that could not be transmitted. Its elements are defined in Table 22 and Table 19.

Table 22 – ApplyResult DataType Structure

Name	Type	Description
ApplyResult	structure	This structure is returned in case of errors. No result data are returned. Further calls with the same TransferId are not possible.
status	Int32	0 – OK – the transferIncidents field may include individual Variables that failed -1 – E_InvalidId – the specified EditContext is unknown
transferIncidents	TransferIncident[]	If the service returns normally and the TransferIncidents list is empty, all changes have been applied and the edited values are cleared. Otherwise, the list contains Variables that could not be transferred successfully. The edited Values are preserved.
contextNodeId	NodeId	The NodeId returned from RegisterNodesById or RegisterNodesByRelativePath.
statusCode	StatusCode	OPC UA StatusCode as defined in IEC 62541-4 and in IEC 62541-8.
diagnostics	DiagnosticInfo	Diagnostic information. This parameter is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in the processing of the request. The DiagnosticInfo type is defined in IEC 62541-4.

9.3.7 Reset Method

Clears all modified values in the EditContext.

The signature of this Method is specified below. Table 23 and Table 24 specify the arguments and AddressSpace representation, respectively.

Signature

```
Reset(
  [in]  String      EditContextId,
  [out] Int32       ResetStatus);
```

Table 23 – Reset Method Arguments

Argument	Description
EditContextId	Identifier of an EditContext that was previously acquired with a GetEditContext call.
ResetStatus	0 – OK -1 – E_InvalidId – the specified EditContext is unknown

Table 24 – Reset Method AddressSpace Definition

Attribute	Value				
BrowseName	Reset				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.3.8 Discard Method

This Method releases the EditContext. Any modified values that have not been applied are lost.

The signature of this Method is specified below. Table 25 and Table 26 specify the arguments and AddressSpace representation, respectively.

Signature

```
Discard(
  [in]  String     EditContextId,
  [out] Int32      DiscardStatus);
```

Table 25 – Discard Method Arguments

Argument	Description
EditContextId	Identifier of an EditContext that was previously acquired with a GetEditContext call.
DiscardStatus	0 – OK -1 – E_InvalidId – the specified EditContext is unknown -2 – E_ChildExists – the specified EditContext cannot be discarded, because a child instance exists.

Table 26 – Discard Method AddressSpace Definition

Attribute	Value				
BrowseName	Discard				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.4 Direct Device Access

9.4.1 General

DirectDeviceAccess provides the means to communicate with a device in the Device Topology. It will be used by UIPs for operations that cannot or at least not easily be performed through Information Model access. Use cases include the transmission of large data buckets from or to the device, for example, historical data or firmware. It is generally assumed that directly accessed data will not be reflected in the Information Model as well. DirectDeviceAccess shall not influence the structure and the data integrity of the Information Model. FDI Servers may be restrictive about when they enable the use of these Methods.

A DirectDeviceAccess Object shall exist for every Device where the FDI Server allows direct access via a UIP.

The following behaviour applies when using DirectDeviceAccess:

- Only one FDI Client can use DirectDeviceAccess at a given time.
- DirectDeviceAccess requires a lock. If the Device has not been locked, the request will be rejected.
- Due to the lock, no write operations and no Method invocations from other FDI Clients are permitted during DirectDeviceAccess. This includes the execution of Actions.
- If Attribute values of Parameters might have changed due to DirectDeviceAccess, the UIP shall set the InvalidateCache in the EndDirectAccess argument to True.

9.4.2 DirectDeviceAccess Type

The DirectDeviceAccessType provides the Methods needed to open and close a connection and to transfer data. Figure 16 shows the DirectDeviceAccessType definition. It is formally defined in Table 27.

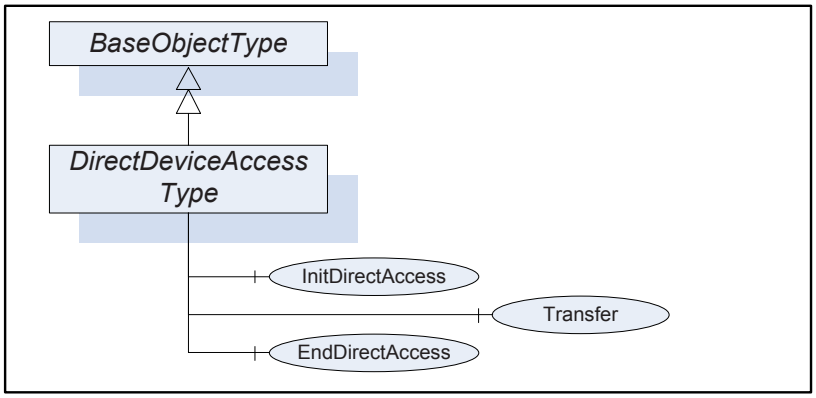

Figure 16 – DirectDeviceAccessType

Table 27 – DirectDeviceAccessType Definition

Attribute	**Value**				
BrowseName	DirectDeviceAccessType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the BaseObjectType defined in IEC 62541-5.					
HasComponent	Method	InitDirectAccess			Mandatory
HasComponent	Method	Transfer			Mandatory
HasComponent	Method	EndDirectAccess			Mandatory

The DirectDeviceAccessType and each instance of this Type share the same Methods. The NodeId of these Methods will be fixed and defined in this standard. FDI Clients therefore do not have to browse for these Methods. They can use the fixed NodeId as the MethodId of the Call Service.

The OPC UA StatusCode Bad_MethodInvalid shall be returned from the Call Service for elements where the DirectDeviceAccess Methods are not supported.

9.4.3 DirectDeviceAccess Object

The support of DirectDeviceAccess for an Object is declared by aggregating an instance of the DirectDeviceAccess Type as illustrated in Figure 17.

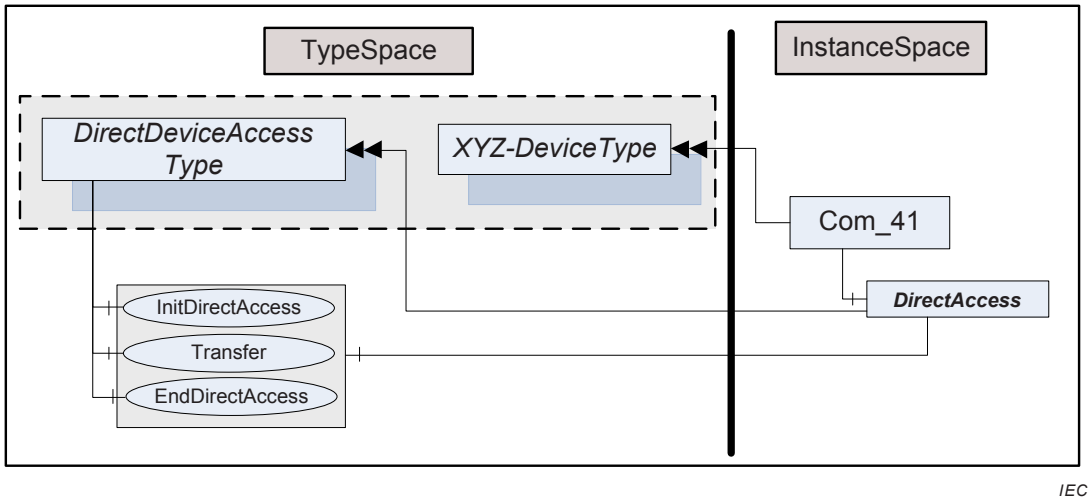

Figure 17 – DirectDeviceAccess instance

This Object is used as container for the DirectDeviceAccess Methods and shall have the BrowseName DirectAccess. It is formally defined in Table 28. HasComponent is used to reference from a TopologyElement (for example, a Device) to its "DirectDeviceAccess" Object.

The DirectDeviceAccessType and each DirectAccess Object share the same Methods.

Table 28 – DirectDeviceAccess Instance Definition

Attribute	Value				
BrowseName	DirectAccess				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasTypeDefinition	ObjectType	DirectDeviceAccessType	Defined in 9.4.2.		

9.4.4 InitDirectAccess Method

InitDirectAccess opens a logic communication channel. The Device to access is specified via the ObjectId argument of the Call Service.

It is up to the FDI Server whether this Method already opens a connection to the physical device.

The signature of this Method is specified below. Table 29 and Table 30 specify the arguments and AddressSpace representation, respectively.

Signature

```
InitDirectAccess(
  [in]  String     Context,
  [out] Int32      InitDirectAccessError);
```

Table 29 – InitDirectAccess Method Arguments

Argument	Description
Context	A string used to provide context information about the current activity going on in the FDI Client/UIP.
InitDirectAccessError	0 – OK -1 – E_NotSupported – the device can not be directly accessed -2 – E_LockRequired – the element is not locked as required -3 – E_InvalidState – the device is already in DirectAccess mode

Table 30 – InitDirectAccess Method AddressSpace Definition

Attribute	Value				
BrowseName	InitDirectAccess				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.4.5 EndDirectAccess Method

EndDirectAccess ends direct access. The directly accessed Device is specified via the ObjectId argument of the Call Service.

The signature of this Method is specified below. Table 31 and Table 32 specify the arguments and AddressSpace representation, respectively.

Signature

```
EndDirectAccess(
  [in]  Boolean      InvalidateCache,
  [out] Int32        EndDirectAccessError);
```

Table 31 – EndDirectAccess Method Arguments

Argument	Description
InvalidateCache	If True, the FDI Server will invalidate any cached values for Device Parameters. This means that these Parameters will be re-read before they are used again.
EndDirectAccessError	0 – OK -1 – E_InvalidState – the device is not in DirectAccess mode.

Table 32 – EndDirectAccess Method AddressSpace Definition

Attribute	Value				
BrowseName	EndDirectAccess				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

9.4.6 Transfer Method

Transfer is used to transfer data to and from the Device. The format of send or receive data is protocol specific.

The signature of this Method is specified below. Table 33 and Table 34 specify the arguments and AddressSpace representation, respectively.

Signature

```
Transfer(
  [in]  String        SendData,
  [out] String        ReceiveData,
  [out] Int32         TransferError);
```

Table 33 – Transfer Method Arguments

Argument	Description
SendData	XML document based on the TransferSendDataType as specified in the communication profile-specific XML schema. See IEC 62769-4:2015, Annex F and IEC 62769-10*-*.
ReceiveData	XML document based on the TransferResultDataType as specified in the communication profile-specific XML schema. See IEC 62769-4:2015, Annex F and IEC 62769-10*-*.
TransferError	0 – OK -1 – E_InvalidState – the device is not in DirectAccess mode.

Table 34 – Transfer Method AddressSpace Definition

Attribute	Value				
BrowseName	Transfer				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Mandatory

10 Parameter Types

10.1 General

IEC 62541-100 defines a Parameter as “a variable of the Device that can be used for configuration, monitoring or control purposes. In the Information Model it is synonymous to an OPC UA DataVariable.”

When discussing Parameter Types we have to consider DataType and VariableType.

Each OPC UA DataVariable (Parameter) has a Value Attribute, which is of a certain DataType. OPC UA has already defined a set of built-in DataTypes, which are sufficient for the majority of types needed for FDI.

VariableTypes represent the type definition of (Data)Variables. Such a type definition typically defines certain behaviour as well as mandatory or optional Properties. The HasTypeDefinition Reference is used to define the VariableType for a Variable. OPC UA already defines a set of VariableTypes as illustrated in Figure 18. BaseDataVariableType is the base type and can be used if no more specialized type information is available (see IEC 62541-5). AnalogItemType and DiscreteItemTypes (see IEC 62541-8) are more concrete types.

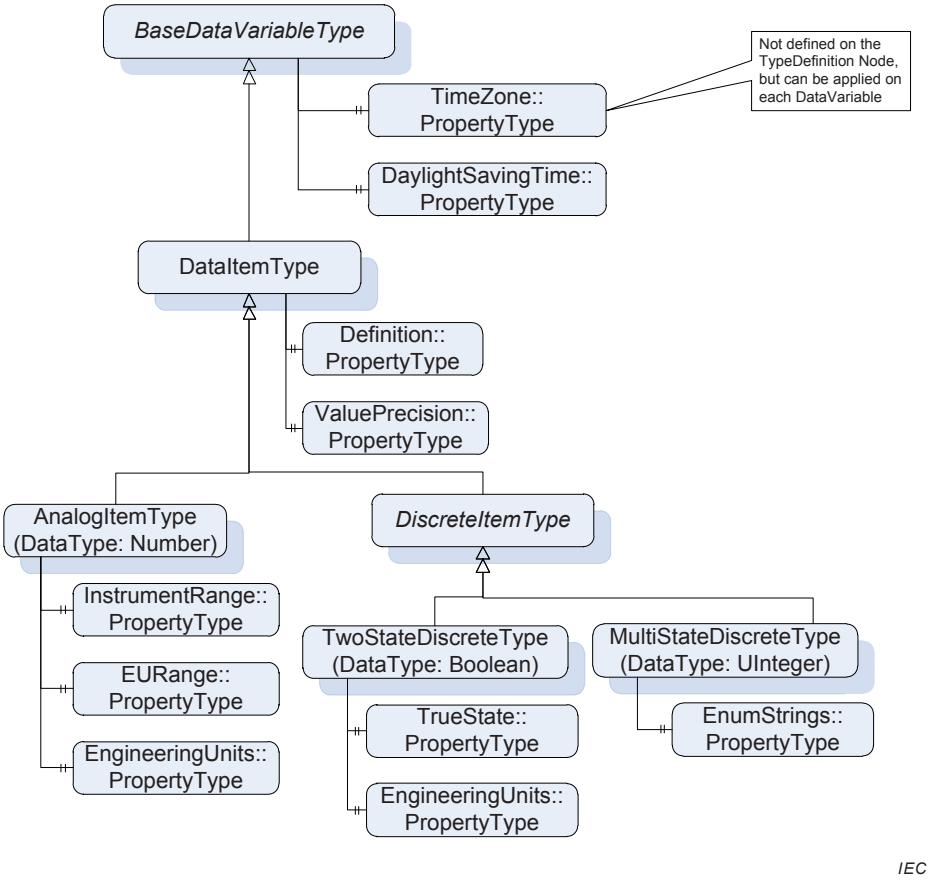

Figure 18 – OPC UA VariableTypes including OPC UA DataAccess

The mapping of EDD Data Types to OPC UA DataTypes and VariableTypes is specified in 15.5.

10.2 ScalingFactor Property

The Value Attribute of Variables contains the raw value returned from the device. However, Servers can expose the Property ScalingFactor. It is suggested, that the (raw) value is multiplied by this factor before being displayed.

The Property shall be aggregated by each Variable that it applies to. It is formally defined in Table 35.

Table 35 – ScalingFactor Property Definition

Name	DataType	Description
ScalingFactor	Double	This Property specifies the scaling factor to be used when displaying the Variable value.

10.3 Min_Max_Values Property

This Property specifies one or more ranges to which a Variable value shall be set. If there are multiple ranges they shall not overlap. A “Null” for the MIN_Value or the MAX_Value indicates that this boundary is not limited.

The Property shall be aggregated by each Variable that it applies to. It is formally defined in Table 35.

Table 36 – Min_Max_Values Property Definition

Name	DataType	Description
Min_Max_Values	Variant_Range[]	This Property specifies the range or ranges to which a Variable Value shall be set.

The Variant_Range structure specifies a single range (a MIN_Value and a MAX_Value). The BaseDataType is used, because such a range can be applied to Variables of different DataTypes – in particular integer, floating point, date and duration. The actual datatype used has to match the DataType of the Variable value. It can also be Null which means that this boundary is not defined.

Its elements are defined in Table 37.

Table 37 – Variant_Range DataType Structure

Name	Type	Description
Variant_Range	structure	
MIN_Value	BaseDataType	Specifies the upper bound of values to which a Variable shall be set. Null indicates that MIN_Value has no limit.
MAX_Value	BaseDataType	Specifies the lower bound of values to which a Variable shall be set. Null indicates that MAX_Value has no limit.

Its representation in the AddressSpace is defined in Table 38.

Table 38 – Variant_Range Definition

Attributes	Value
BrowseName	Variant_Range

11 FDI StatusCodes

Clause 11 defines OPC UA StatusCodes that are specific to FDI Servers.

The general structure of the StatusCode is specified in IEC 62541-4. It includes a set of common operational result codes which also apply to FDI.

Table 39 contains Good (success) codes that are specifically defined for FDI.

Table 39 – Good operation level result codes

Symbolic Id	Description
Good_Edited	This status applies to Variable values that are part of an EditContext (see 9.3). It is returned with values that are read or received in a DataChangeNotification from a Subscription. It defines that it is an edited value that has not been transferred from the EditContext to the Device.
Good_ PostActionFailed	The value of a Variable was successfully read or written but one of the post actions failed.
Good_ DependentValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3).It is delivered with a dominant Variable value when any of its dependent Variables was changed and not yet applied.

12 Specialized topology elements

Devices and other topology elements can be specialized using the modelling elements defined in this standard or in IEC 62541-100. No specific ObjectType is needed. A

Communication Device for instance will have the CommunicationServices capability (Communication Devices and CommunicationServices are specified in IEC 62769-7).

Other specializations are possible applying the same techniques, for example, a Modular Communication Device.

See IEC 62541-100 for the definition of Block Device and Modular Device.

13 Auditing

13.1 General

Auditing is a requirement in many systems. It provides a means for tracking activities that occur as part of the normal operation of the system. It also provides a means for tracking abnormal behaviour. It is also a requirement from a security standpoint.

When an audit trail is maintained by the FDI Server, all audit trail records related to services invoked by FDI Clients will be implicitly created. In addition, FDI Clients have means for providing additional audit context information as specified in 13.2 and 13.3.

FDI Servers shall generate AuditEvents as specified in IEC 62541-4 and IEC 62541-5. These standards define AuditEvents for the following OPC UA Services:

- Secure Channel Services
- Session Services
- NodeManagement Service Set (AddNode, DeleteNode)
- Write Service
- Method Service (“Call”)

No AuditEvents are defined for reading and for subscriptions.

The FDI Server generates AuditEvents using the standard OPC UA event mechanism. This allows an external OPC UA Client to subscribe to and store the audit entries in a log file or other storage location.

13.2 FDI Client-provided context information

FDI Clients have various means for providing context information for the purpose of auditing:

- the ClientDescription and SessionName passed by the FDI Client when creating the Session,
- the context text from the Methods to initialise Lock or Direct Device Access.

When an AuditUpdateMethodEvent is generated for the EnterLock and InitDirectAccess Methods, the audit context information will be used for the Message field of this event.

13.3 LogAuditTrailMessage Method

LogAuditTrailMessage is used by the FDI Client to insert information about the current activity going on in the FDI Client into the audit trail of the FDI Server. Since it is a Method, it will be also inserted into the stream of AuditEvents. The message will be timestamped in the FDI Server.

This Method shall be a component of the Server Object. The ObjectId parameter of the Call Service shall be the NodeId of the Server Object.

FDI Clients do not have to browse for the LogAuditTrailMessage Method. Rather they can use the well known NodeId of the Method declaration as the MethodId of the Call Service.

The signature of this Method is specified below. Table 40 and Table 41 specify the arguments and AddressSpace representation, respectively.

Signature

```
LogAuditTrailMessage(
  [in]  String       Message);
```

Table 40 – LogAuditTrailMessage Method Arguments

Argument	Description
Message	Free text describing the context.

Table 41 – LogAuditTrailMessage Method AddressSpace Definition

Attribute	Value				
BrowseName	LogAuditTrailMessage				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Mandatory

When an AuditUpdateMethodEvent is generated for the LogAuditTrailMessage Method, the Message argument will be used for the Message field of this event.

14 FDI Server Version

The FDI Technology Version supported by an FDI Server is exposed via an FDI-specific Property. The version exposed with this Property applies to

- the Information Model, and
- the XML Schema as used for UIDs and Actions.

The Property shall be aggregated by the OPC UA Server Object. It is formally defined in Table 42.

Table 42 – FDIServerVersion Property Definition

Name	DataType	Description
FDIServerVersion	String	This Property specifies the FDI Technology Version that this FDI Server supports. The syntax of the string is as defined in IEC 62769-4.

15 Mapping FDI Package information to the FDI Information Model

15.1 General

Clause 15 defines the mapping of EDDL and other FDI Package Information to the FDI Information Model and the underlying OPC UA and OPC UA Devices information models, respectively.

The OPC UA Object Model provides a standard way for Servers to represent Objects to Clients. In order to meet this objective, the OPC UA Object Model allows the definition of Objects in terms of Variables and Methods. It also allows relationships to other Objects to be expressed.

On the other hand, EDDL defines a set of language constructs that are used to describe industrial field devices. Each construct supports its own set of attributes and references. EDD information adds semantic contents to the raw data values read from and written to the field devices.

The primary objective of the EDDL-OPC UA information model is to describe the correspondence between the OPC UA Object Model elements and the EDDL elements when an EDD is used to populate the FDI Server with Objects.

Completely meeting this objective means dealing both with enhanced data access through OPC UA and with UI interactions between OPC UA clients and servers.

15.2 Localization

15.2.1 Localized text

In various definitions, EDDs may contain information in multiple language variants. Examples are the LABEL and HELP attributes. The server has to select the proper language for each client as follows: When creating an OPC UA Session, the OPC UA Client passes the locale(s) that it requests to be used for all services within this Session. If the Server does not support any of the requested locales it chooses an appropriate default locale.

15.2.2 Engineering units

Variables with a UNIT CODE are represented in OPC UA as AnalogItem Variables. The UNIT CODE is exposed via the EngineeringUnit Property. Changing the EngineeringUnit will cause all EDD Variables that depend on the associated UNIT CODE to be recalculated. As a result the OPC UA Variable values will be set as well.

Changing the EngineeringUnit will affect all Clients.

15.3 Device

15.3.1 General

Subclause 15.3 specifies the mapping of FDI Package elements to Device Types and Device Instances.

Devices may also have Blocks (see 15.4).

15.3.2 Mapping to Attributes to a specific DeviceType Node

The BrowseName and NodeId Attributes are vendor specific.

The DisplayName Attribute is created from the Package Catalog: DeviceType.Name.

The Description Attribute of a Device is information that serves to further identify, manage, locate, and/or explain the device whose contents are defined by the user. For purely block-oriented devices this will appear only on blocks. For example, in HART the MESSAGE variable can be used here.

15.3.3 Mapping to Properties

Type and instances share the same Properties.

Some of the Properties are created from information in the Package Catalog. Table 43 specifies the mapping of Properties to Package Catalog elements.

Table 43 – DeviceType Property Mapping

Property	Package Catalog element
Manufacturer	ManufacturerName
Model	ListOfDeviceTypes[i].ListOfInterfaces[j].DeviceModel.
DeviceRevision	ListOfDeviceTypes[i]. ListOfInterfaces[j].Version.
FDITechnologyVersion	FDITechnologyVersion.

The SerialNumber, RevisionCounter, SoftwareRevision, and HardwareRevision Properties correspond to the value of the protocol-specific “serial number”, “revision counter”, “software revision”, and “hardware revision” Parameters, respectively.

The DeviceManual Property is an empty string. Documents can be found in the attachment set.

If a Picture element is available in the FDI Package it can be mapped to the OPC UA Icon Property (see IEC 62541-3). If multiple resolutions are available the host has to choose.

15.3.4 Mapping to ParameterSet

Some devices are strictly block-oriented with no kinds of Variables on the device-level. Mapping an EDD for these devices will result in an empty ParameterSet. When VARIABLE, VALUE_ARRAY or LIST items exist on the device-level, the resulting Parameters are kept in the “ParameterSet” as a flat list of Parameters. FunctionalGroups reflect the structure of the device menus defined in the device’s EDD.

A ParameterSet with all Parameters exists on the Type, the offline and the online instances.

See 15.5 on how EDDL attributes are used to create Parameter nodes.

15.3.5 Mapping to Functional Groups

The top-level Functional Groups that are referenced directly from the Device Object correspond to the root MENU items as defined in IEC 61804-4. Naming conventions are used to differentiate between Functional Groups for handheld and for PC-based applications.

There are no Functional Groups on the DeviceType.

15.3.6 Mapping to DeviceTypeImage

The Variables in the DeviceTypeImage folder are created from the Package Catalog: DeviceTypes[i].ListOfDeviceImages. BrowseName and DisplayName represent the file name from the Relationship Type.

15.3.7 Mapping to Documentation

The Variables in the Documentation folder are created from the Package Catalog: DeviceTypes[i].ListOfDocuments. BrowseName and DisplayName represent the file name from the Relationship Type.

15.3.8 Mapping to ProtocolSupport

The Variables in the ProtocolSupport folder are created from the Package Catalog: DeviceTypes[i].ListOfInterfaces[j].ListOfCommunicationProfileSupportFiles. BrowseName and DisplayName represent the file name from the Relationship Type.

15.3.9 Mapping to ImageSet

The ImageSet contains Variable Nodes for all images from the EDD that are needed for UIDs.

15.3.10 Mapping to ActionSet

The ActionSet references OPC UA Objects that represent all EDD Methods except for the abort and the action methods as defined in IEC 61804-3. See 15.7 on how EDDL attributes are used to create Action Nodes

15.3.11 Mapping to MethodSet

The MethodSet (inherited from IEC 62541-100) is not used by the FDI.

15.4 Block

15.4.1 General

Subclause 15.4 specifies the mapping of EDDL elements to Block Types and instances.

EDDL supports the definition of devices that are block-oriented, and devices that are non-block oriented. When blocks (specifically, Block_A) exist in an EDD, the resulting device shall be modelled as block-oriented Device as specified in IEC 62541-100. The Block node instances are kept in the Blocks component. A Device that does not support EDDL defined Blocks will not have the Blocks component.

The SupportedTypes folder in the Blocks component of a DeviceType references all BlockTypes that may appear in a Device Instance. The Blocks component in a Device Instance references instantiated Blocks.

15.4.2 Mapping to Attributes

The BrowseName and DisplayName correspond to the EDD Identifier and LABEL Attribute of the corresponding EDDL Block, respectively.

The NodeId is vendor specific.

The Help attribute of the corresponding EDDL Block is mapped to the Description attribute of the Block instance node. Bad_AttributeIdInvalid shall be used if EDD contains no Help.

15.4.3 Mapping to ParameterSet

All VARIABLE, VALUE_ARRAY, LIST items specified for a Block are used as Parameters in the “ParameterSet”. A ParameterSet with all Parameters exist on the Type, the offline and the online instances.

See 15.5 on how EDDL attributes are used to create Parameter nodes.

15.4.4 Mapping to Functional Groups

A Block may have FunctionalGroups that expose its Parameters in an organized fashion, reflecting the structure of the block menus defined in the device’s EDD. The top-level Functional Groups that are referenced directly from the Block Object correspond to the root MENU items as defined in IEC 61804-4. Naming conventions are used to differentiate between Functional Groups for handheld and for PC-based applications

The BrowseName of a FunctionalGroup is the EDD identifier of the corresponding EDDL MENU or COLLECTION.

There are no Functional Groups on the BlockType.

15.4.5 Mapping to ActionSet

The ActionSet references OPC UA Objects that represent all EDD Methods defined for a specific BLOCK except for the abort and the action methods as defined in IEC 61804-3. See 15.7 on how EDDL attributes are used to create Action Nodes

15.4.6 Mapping to MethodSet

The MethodSet (inherited from IEC 62541-100) is not used by the FDI.

15.4.7 Instantiation rules

The BrowseName of a Block is generated by the FDI Server from the EDD identifier of the corresponding EDDL BLOCK_A by adding a numeric suffix that the OPC UA server generates in order to make the BrowseName unique. For example, __analog_input_0, __analog_input_1, __pid_control_0.

The DisplayName of a BLOCK_B Block is the LABEL attribute.

The DisplayName of a BLOCK_A Block is defined in the protocol annex.

The Description of a Block is the HELP attribute of the corresponding EDDL BLOCK. Bad_AttributeIdInvalid shall be used if the EDD contains no Help.

15.5 Parameter

15.5.1 General

EDDL Parameters (for devices or blocks) are mapped to OPC UA Variables. The VariableType used may be of any sub-type of the abstract BaseVariableType. In most cases they will be mapped to the VariableTypes defined in IEC 62541-8, for example, DataItemType, AnalogItemType or DiscreteItemType. This standard includes additional VariableTypes for more sophisticated EDDL types. See Table 45 for the mapping of EDDL types.

The BrowseName of a Parameter is the EDD identifier of the corresponding EDDL VARIABLE, RECORD or VALUE_ARRAY.

The DisplayName of a Parameter is the LABEL attribute of the corresponding EDDL VARIABLE, RECORD or VALUE_ARRAY.

The Description of a Parameter is the HELP attribute of the corresponding EDDL VARIABLE, RECORD or VALUE_ARRAY. Bad_AttributeIdInvalid shall be used if the EDD contains no Help.

Parameters have also a set of Attributes that are common to all VariableTypes. Table 44 summarizes the Variable Attributes and describes how they are set from the EDDL description information.

Table 44 – Setting OPC UA Variable Attributes from EDDL variable attributes

<table>
<tr><th>Attributes</th><th>Description</th></tr>
<tr><td>Value</td><td>The most recent value of the Variable that the FDI Server has read from the device.</td></tr>
<tr><td>DataType</td><td>The EDDL data type is translated into an OPC UA standard data type according to Table 45.</td></tr>
<tr><td>ValueRank</td><td>Either set to “Scalar” or – when the Parameter is an array – the NUMBER_OF_ELEMENTS specified for the EDDL VALUE_ARRAY item.</td></tr>
<tr><td>AccessLevel</td><td>The AccessLevel Attribute is set based on the EDDL variable HANDLING attribute according to the following table:
<table>
<tr><th>Field</th><th>Bit</th><th>Description</th></tr>
<tr><td>CurrentRead</td><td>0</td><td>Set if EDDL variable HANDLING is defined as READ. Reset otherwise.</td></tr>
<tr><td>CurrentWrite</td><td>1</td><td>Set if EDDL variable HANDLING is defined as WRITE. Reset otherwise.</td></tr>
</table>
If the HANDLING attribute is missing, the Parameter will be defined as readable and writeable.</td></tr>
<tr><td>UserAccessLevel</td><td>The AccessLevel with possible restrictions based on client identity as defined by the FDI Server.</td></tr>
<tr><td>MinimumSamplingInterval</td><td>The MinimumSamplingInterval Attribute indicates how fast the FDI Server can reasonably sample the value for changes. It is suggested that the FDI Server checks the variable CLASS attribute to differentiate static variables from dynamic variables regarding the sampling interval. Static variables might be sampled only once and then only when the RevisionCounter changes. For static variables the FDI Server can use the MinimumSamplingInterval -1 (indeterminate).</td></tr>
</table>

The Value Attribute is the Parameter value, and – for the online representation – reflects the device data value.

The DataType Attribute is an OPC UA DataType chosen to match the EDDL type and size. Table 45 shows the correspondence between the EDDL types and sizes and the OPC UA VariableTypes and standard DataTypes.

Table 45 – Correspondence between EDDL and OPC UA standard data types

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
Arithmetic			
INTEGER	BaseDataVariableType, AnalogItemType	SByte	When the size specified in EDDL is 1 byte.
		Int16	When the size specified in EDDL is 2 bytes.
		Int32	When the size specified in EDDL is 3 or 4 bytes.
		Int64	When the size specified in EDDL is 5, 6, 7 or 8 bytes.
UNSIGNED_INTEGER	BaseDataVariableType, AnalogItemType	Byte	When the size specified in EDDL is 1 byte.
		UInt16	When the size specified in EDDL is 2 bytes.
		UInt32	When the size specified in EDDL is 3 or 4 bytes.
		UInt64	When the size specified in EDDL is 5, 6, 7 or 8 bytes.
DOUBLE	BaseDataVariableType, AnalogItemType	Double	
FLOAT	BaseDataVariableType, AnalogItemType	Float	
ENUMERATED	See 15.5.5	Byte	When the size specified in EDDL is 1 byte.
		UInt16	When the size specified in EDDL is 2 bytes.
		UInt32	When the size specified in EDDL is 3 or 4 bytes.
		UInt64	When the size specified in EDDL is 5, 6, 7 or 8 bytes.
BIT_ENUMERATED	See 15.5.6	Byte	When the size specified in EDDL is 1 byte.
		UInt16	When the size specified in EDDL is 2 bytes.
		UInt32	When the size specified in EDDL is 3 or 4 bytes.
		UInt64	When the size specified in EDDL is 5, 6, 7 or 8 bytes.
Date-and-Time			
DATE	BaseDataVariableType	UtcTime (a 64-bit signed integer which represents the number of 100 nanosecond	The data type DATE consists of a calendar date. This data type has special codification in the device. Conversion to UtcTime is necessary when reading it from the device. Conversion back to DATE is necessary when writing it to the device. On write, if invalid data is written, the

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
		intervals since January 1, 1601 (UTC))	service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.
DATE_AND_TIME	BaseDataVariableType	UtcTime	The data type DATE_AND_TIME consists of a calendar date and a time. This data type has special codification in the device. Conversion to UtcTime is necessary when reading it from the device. Conversion back to DATE_AND_TIME is necessary when writing it to the device. On writing if invalid data is written the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.
DURATION	BaseDataVariableType	Duration (a *Double* that defines an interval of time in milliseconds (fractions can be used to define sub-millisecond values))	The DURATION data type is a time difference that consists of a time in milliseconds and an optional day count. This data type has special codification in the device. Conversion to Time is necessary when reading it from the device. Conversion back to DURATION is necessary when writing it to the device. On writing, if invalid data is written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data. The OPC Duration type is limited to approximately 49,7 days, which is less than the theoretical maximum of the EDDL DURATION type. If the DURATION exceeds the OPC maximum the quality should be set to indicate this condition.
TIME	BaseDataVariableType	UtcTime	The TIME data type consists of a time and an optional date. This data type has special codification in the device. Conversion to UtcTime is necessary when reading it from the device. Conversion back to TIME is necessary when writing it to the device. On writing, if invalid data is written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.
TIME_VALUE[4]	BaseDataVariableType	Duration	Number of 1/32 ms.
TIME_VALUE[8]	BaseDataVariableType	UtcTime	The data type TIME_VALUE is used to represent date and time in the required precision for application clock synchronization. This data type has special codification in the device. Conversion to UtcTime is necessary when reading it from the device. Conversion back to TIME_VALUE is necessary when writing it to the device. On writing, if invalid data is written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.
String			
ASCII	BaseDataVariableType	String	Conversion to String is necessary when reading it from the device. Conversion back to ASCII is necessary when writing it to the device. On writing,if invalid characters are written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the string.
BIT_STRING	BaseDataVariableType	ByteString	Conversion to ByteString is necessary when reading it from the device. Conversion back to BIT_STRING is necessary when writing it to the device. On writing, if invalid characters are written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the string.
EUC	BaseDataVariableType	String	Conversion to String is necessary when reading it from the device. Conversion back to EUC is necessary when writing it to the device. On writing, if invalid characters are written, the

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
			service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the string.
PACKED_ASCII	BaseDataVariableType	String	Conversion to String is necessary when reading it from the device. Conversion back to PACKED_ASCII is necessary when writing it to the device. On writing, if invalid characters are written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the string.
PASSWORD	BaseDataVariableType	String	Conversion to String is necessary when reading it from the device. FDI Servers shall allow PASSWORD Variables to be read only when a secure OPC UA channel has been created, i.e., a channel with encryption.
VISIBLE	BaseDataVariableType	String	Conversion to String is necessary when reading it from the device. Conversion back to VISIBLE is necessary when writing it to the device. On writeing,if invalid characters are written, the service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the string.
OCTET	BaseDataVariableType	ByteString	
INDEX	see 15.5.5	String	
BOOLEAN	BaseDataVariableType	Boolean	

As Table 45 shows, EDDL supports a variety of variable types. While that table shows the correspondence between the EDDL data types and the OPC UA VariableType, it does not provide any details on how other EDDL VARIABLE TYPE construct attributes are supported. It does not provide any details on how other OPC UA BaseDataVariableType attributes should be set either. Subclause15.5 is intended to provide details for each EDDL data type.

15.5.2 Private Parameters

The Parameters specified in an FDI Package may be declared private using the PRIVATE Attribute specified in 61804-3. The FDI Server shall create Nodes in the Information Model for the private Parameters but they shall not be browsable. The FDI Server shall return the NodeIds of private Parameters when the name of such a Parameter is passed to TranslateBrowsePathsToNodeIds (the startingNode argument shall be the “ParameterSet” Object). Once the FDI Client has obtained the NodeId all Service requests for private Parameters will be processed in the same way as for public (browsable) Parameters.

An example of private parameters is parameters that should only be modified through an Action. These parameters should not be visible to FDI Clients to prevent direct access. FDI Clients invoke Actions to access these private parameters.

15.5.3 MIN_Value and MAX_Value

If one or more MIN_VALUE and MAX_VALUE attributes are specified for a variable in EDDL they shall be mapped to the Min_Max_Values Properties defined in 10.3.

15.5.4 Engineering units

The EngineeringUnits Property defined in IEC 62541-3 and IEC 62541-8 shall be used:

- If the variable has the CONSTANT_UNIT defined for itself in EDDL. In this case the EngineeringUnits Property keeps the EDD CONSTANT_UNIT variable attribute. The FDI Server shall deal with the fact that CONSTANT_UNIT can be conditional.

- If the variable is involved in an EDD UNIT relation it is the FDI Server's responsibility to implement the unit code update mechanism. An EDD UNIT relation specifies a reference to a variable holding a unit code and a list of dependent variables. When the variable holding the unit code is modified, the list of effected variables using that unit code shall be refreshed. When an effected variable is displayed, the value of its unit code shall also be displayed.

The standard OPC UA notifications can be used to report to the FDI Clients that the unit code changed.

15.5.5 Enumerated Parameters

An OPC UA DataVariable with the OPC UA DataType Enumeration is used for each enumerator in the EDDL enumerated variable definition.

The Value Attribute of the DataVariable is the numeric value of the state, and corresponds to the value attribute of the EDDL ENUMERATED TYPE.

The ValueAsText Property of the DataVariable exposes the display value of the state, which corresponds to the description attribute of the EDDL ENUMERATED TYPE.

The EnumValues Property of the DataVariable contains the complete list of enumerations, i.e., a table where each element is a structure consisting of EDDL ENUMERATED TYPE attributes "value", "description", and "help". If no help attribute exists in the EDD, the value of the description attribute will also be used for this purpose.

15.5.6 Bit-enumerated Parameters

A DataVariable of OPC UA OptionSet VariableType is used for each EDDL BIT ENUMERATED VARIABLE definition. The OPC UA DataType is an array of Boolean where one Boolean is used per bit in the EDDL BIT_ENUMERATED VARIABLE definition.

The Boolean array of the DataVariable reflects the status of all bits. TRUE is used when a bit is set and FALSE when a bit is reset.

The EnumValues Property of the DataVariable contains the complete list of bit enumerations, i.e., a table where each element is a structure consisting of EDDL BIT ENUMERATED TYPE attributes "bit position", "description", and "help". If no help attribute exists in the EDD, the value of the description attribute will also be used for this purpose.

15.5.7 Representation of records

A complex DataVariable is used to represent EDDL RECORD Parameters. The root DataVariable represents the record itself. It will have component DataVariables that represent the EDDL RECORD MEMBERS. (The MEMBERS of an EDDL RECORD are defined in EDDL by means of a reference to an EDDL VARIABLE.).

See Figure 19 for an example of how records are represented in the OPC UA AddressSpace.

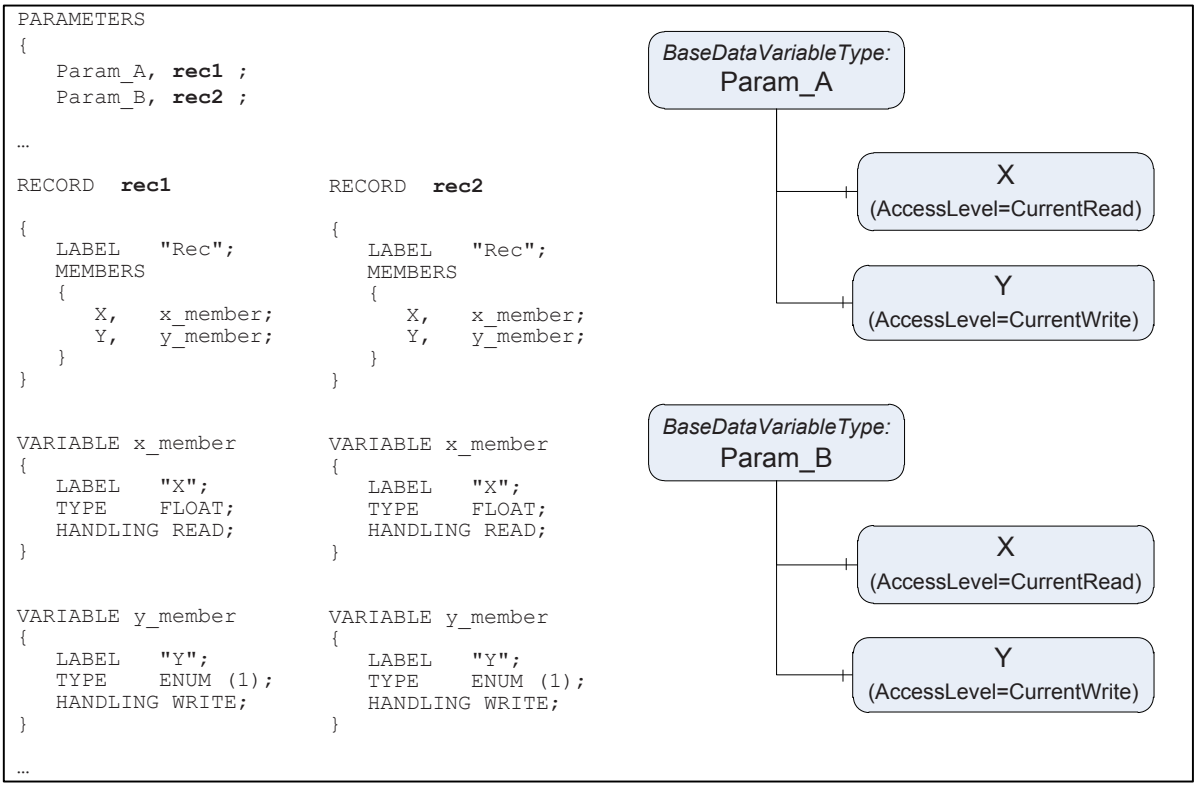

Figure 19 – Example: Complex variable representing a RECORD

BrowseName and DisplayName of the root DataVariable are set to the EDD identifier of the EDDL VARIABLE that implements this RECORD type. The DataType Attribute of the “root” DataVariable is BaseDataType. The ValueRank Attribute is used to specify that the value contains an array. The Value Attribute represents the values of all members in the order as defined for the RECORD. According to the example in Figure 19, the first variant will contain a floating point value and the second will be a numeric value representing the Enumeration.

For each component DataVariable that represents an EDDL RECORD MEMBER:

- The BrowseName is the identifier of the corresponding EDDL VARIABLE.
- The DisplayName is the LABEL attribute of the corresponding EDDL VARIABLE.
- The Description is the HELP attribute of the corresponding EDDL VARIABLE. Bad_AttributeIdInvalid shall be used if the EDD contains no Help.
- The AccessLevel is derived from the HANDLING attribute. Readable and Writeable shall be used if the EDD contains no HANDLING attribute.

15.5.8 Representation of arrays, and lists of Parameters with simple data types

A single DataVariable will represent an EDDL VALUE_ARRAY or LIST item when the data type of the referenced array element has a simple data type.

The OPC UA DataVariable Attributes are set as follows:

- DataType is set to the type of the array element (see Table 45 for the data type mapping).
- The ValueRank Attribute is used to specify that the value contains an array. In case of an EDDL VALUE_ARRAY the number of elements is exposed. In the case of an EDDL LIST the number of elements is unspecific since the size can change dynamically.

15.5.9 Representation of values arrays, and lists of RECORD Parameters

Value arrays or lists of non-simple data types will be represented as OPC UA array of complex variables. Figure 20 shows the EDDL sample code of a VALUE_ARRAY of RECORDs and the corresponding complex variable in the OPC UA AddressSpace.

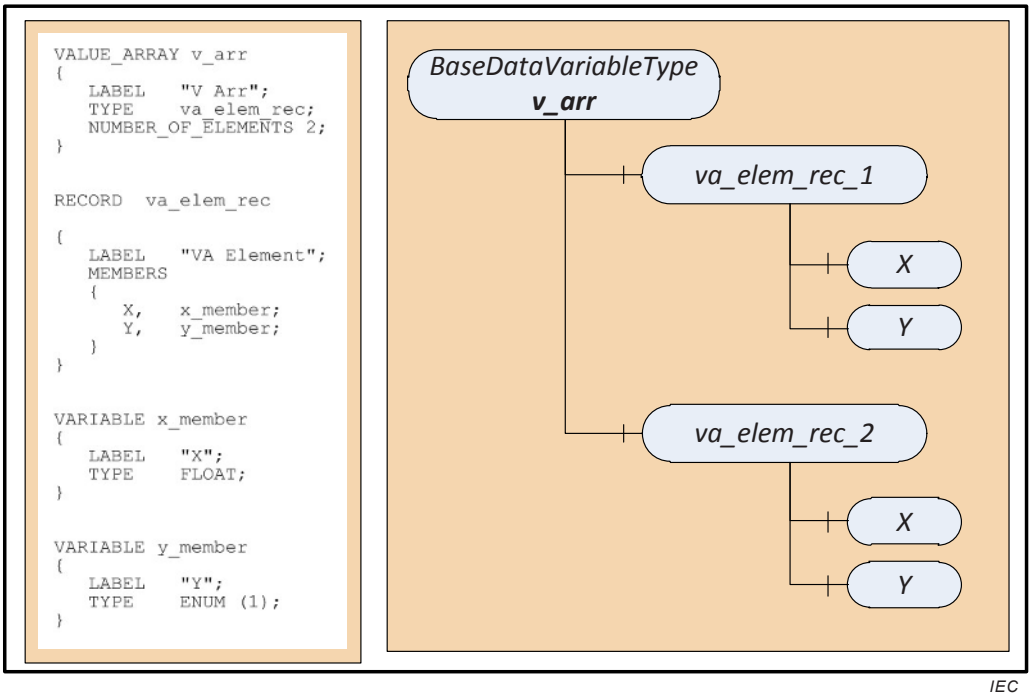

Figure 20 – Complex variable representing a VALUE_ARRAY of RECORDs

In the FDI Server AddressSpace a complex DataVariable is used to represent the entire VALUE_ARRAY. Each VALUE_ARRAY element, which is in fact a RECORD, is represented as a component complex DataVariable. The RECORD MEMBERS are also represented as component DataVariables. The FDI Client refers to the X member of the first record in the array through the BrowseName v_arr.va_elem_1.x_member. Note that the index always begins with '1'.

The DataType Attribute of all complex root DataVariables is BaseDataType. The ValueRank Attribute is used to specify that the value contains an array. The Value Attribute represents all VALUE_ARRAY entries. The first element corresponds to the first array entry and so on. Each element in turn contains an array. This may either be an array of simple types or an array of BaseDataType. A RECORD is always represented as an array of BaseDataType.

15.5.10 Representation of COLLECTION and REFERENCE ARRAY

The EDDL constructs COLLECTION and REFERENCE ARRAY are mapped to Functional Group Objects (see 15.6) when used for grouping only.

15.5.11 SCALING_FACTOR

If the EDD includes a SCALING_FACTOR attribute for an EDD VARIABLE it shall be mapped as follows.

- The Parameter value and associated Properties (like Ranges) shall be exposed in raw format (not scaled by the FDI Server). Any scaling is responsibility of the FDI Client (or UIP, respectively).

- The SCALING_FACTOR shall be provided in a Property with the BrowseName “ScalingFactor”. The DataType shall be “Double”. If the SCALING_FACTOR in the EDD is an expression, this expression will be computed by the EDD Interpreter.
- The SCALING_FACTOR is also included in UID documents.

15.6 Functional Groups

FunctionalGroups are used to group Variables (EDDL parameters) or Actions (EDDL methods). They have a recursive definition, that is, FunctionalGroups may reference other FunctionalGroups.

Both Devices and Blocks may have FunctionalGroups. EDDL MENUs, EDDL BLOCK_A PARAMETERS, COLLECTIONs, and REFERENCE ARRAYS are the building blocks for FunctionalGroups.The proper attributes of these EDDL constructs control which elements are referenced by the Functional Group.

- VARIABLE, RECORD, or VALUE_ARRAY will cause a reference to the corresponding FDI Parameter.
- METHODs will be used to organise FDI Action Objects.
- Elements that represent one of the bulding blocks again will cause a reference to a subordinate Functional Group.

Root MENUs as defined in IEC 61804-4 will be top-level Functional Groups. The FDI Server generates the whole hierarchy of FunctionalGroups by browsing the EDDL MENU and their ITEMS attribute definitions. The FunctionalGroups reproduce the same structure of the MENUs as they are defined in EDD for the Device or Block.

All FunctionalGroups are instantiated in both the offline and the online representation of a Device. Naming conventions defined in IEC 61804-4 can be used to determine if a FunctionalGroup is relevant for offline or online.

The BrowseName Attribute of a FunctionalGroup is the EDD identifier of the menu or the instance identifier of the block. The naming conventions defined in IEC 61804-4 can be used to determine if a FunctionalGroup is relevant for PC or handheld.

The DisplayName Attribute of a FunctionalGroup is the LABEL attribute of the corresponding building block.

The Description Attribute of a FunctionalGroup is the value of the HELP attribute of the corresponding building block. Bad_AttributeIdInvalid shall be used if the EDD contains no Help.

FunctionalGroups exposing standard sets of Parameters such as diagnostic information will be created for an EDD, which comply with the menu conventions for PC-based application as defined in IEC 61804-4.

15.7 AXIS elements in UIDs

AXIS elements are not represented as instances in the FDI information model. However, an EDDL AXIS has writable VIEW_MIN/VIEW_MAX attributes. They shall be modified by the FDI Client to inform the server about the zooming, scrolling or positioning.

To enable this modification, the FDI Server shall create variable nodes for each of these attributes in the Information Model. They need not be browsable. The NodeIds of these nodes are included into the UID document as described in the UID schema.

15.8 Actions

FDI Actions are used to represent EDD METHODs. Only EDD METHODs that are to be called by the FDI Client as part of a UID or a UIP shall be exposed as FDI Actions. If METHODs defined in IEC 61804-3, which include the pre or post read, edit and write actions are exposed as Actions, they shall not be browsable. The names of these actions will be communicated to the Client as part of a UID document. These names can be passed to TranslateBrowsePathsToNodeIds to get the matching NodeId.

All Actions are instantiated in both the offline and the online representation of a Device.

The BrowseName Attribute of an Action is the EDD identifier of the METHOD.

The DisplayName Attribute of an Action is the LABEL attribute of the EDDL METHOD.

The Description Attribute of an Action is the HELP attribute of the EDDL METHOD. Bad_AttributeIdInvalid shall be used if the EDD contains no Help.

The execution of an Action causes the execution of the EDD Method implementation. This is specified in detail in IEC 62769-2.

The METHODs specified in an FDI Package may be declared private using the PRIVATE Attribute specified in 61804-3. The FDI Server shall create Action Objects in the Information Model for the private METHODs but they shall not be browsable. The FDI Server shall return the NodeIds of private Actions when the name of such an Action is passed to TranslateBrowsePathsToNodeIds (the startingNode argument shall be the “ActionSet” Object). Once the FDI Client has obtained the NodeId, private Actions can be invoked and processed in the same way as public (browsable) Actions.

15.9 UIPs

UIPs are specified in the FDI Package via the SupportedUIPs element. Each UIP is uniquely identified by a UUID.

UIPs are not exposed in the FDI AddressSpace but they can be referenced and accessed using the UUID. This is specified in IEC 62769-2.

15.10 Protocols, Networks and Connection Points

The FDI Server will create specific Protocol Types (sub-type of the ProtocolType defined in IEC 62541-100) for natively supported Protocols. Additional Protocol Types might be created for Communication Servers based on the Protocols they support.

Specific ConnectionPoint Types (sub-types of the ConnectionPointType defined in IEC 62541-100) will be created for natively supported Protocols. Additional Types might be created for Communication Servers based on the Protocols they support.

The types are created as specified in the protocol-specific series of standards IEC 62769-1xx.

Annex A
(normative)

Namespace and Mappings

This appendix defines the numeric identifiers for all of the numeric *NodeIds* defined in this standard. The identifiers are specified in a CSV file with the following syntax:

```
<SymbolName>, <Identifier>, <NodeClass>
```

Where the *SymbolName* is either the *BrowseName* of a *Type Node* or the *BrowsePath* for an *Instance Node* that appears in the specification and the *Identifier* is the numeric value for the *NodeId*.

The *BrowsePath* for an *Instance Node* is constructed by appending the *BrowseName* of the instance *Node* to the *BrowseName* for the containing instance or type. An underscore character is used to separate each *BrowseName* in the path.

The *NamespaceUri* http://fdi-cooperation.com/OpcUa/FDI5/ is applied to *NodeIds* defined here.

The CSV released with this version of the standard can be found here:
http://www.fdi-cooperation.com/tl_files/Specification/1.0/Schemas/Opc.Ua.Fdi5.NodeIds.csv

An electronic version of the complete Information Model defined in this standard is also provided. It follows the XML Information Model Schema syntax defined in IEC 62541-6.

The Information Model Schema released with this version of the standard can be found here:
http://www.fdi-cooperation.com/tl_files/Specification/1.0/Schemas/Opc.Ua.Fdi5.NodeSet2

Bibliography

IEC TR 62541-1, *OPC unified architecture – Part 1: Overview and Concepts*

IEC 62541-7, *OPC unified architecture – Part 7: Profiles*

FDI-2021, *FDI Project Technical Specification – Part 1: Overview*
<available at www.fdi-cooperation.com>

FDI-2022, *FDI Project Technical Specification – Part 2: FDI Client*
<available at www.fdi-cooperation.com>

FDI-2023, *FDI Project Technical Specification – Part 3: FDI Server*
<available at www.fdi-cooperation.com>

FDI-2024, *FDI Project Technical Specification – Part 4: FDI Packages*
<available at www.fdi-cooperation.com>

FDI-2025, *FDI Project Technical Specification – Part 5: FDI Information Model*
<available at www.fdi-cooperation.com>

FDI-2026, *FDI Project Technical Specification – Part 6: FDI Technology Mapping*
<available at www.fdi-cooperation.com>

FDI-2027, *FDI Project Technical Specification – Part 7: FDI Communication Devices*
<available at www.fdi-cooperation.com>

NAMUR NE107, *Self-Monitoring and Diagnosis of Field Devices*
<available at www.namur.de >

———————

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

bsi.

...making excellence a habit.™