IEC 62769-5

Edition 3.0 2023-04
REDLINE VERSION

INTERNATIONAL STANDARD

colour inside

Field device integration (FDI®) –
Part 5: FDI Information Model

IEC 62769-5:2023-04 RLV(en)

IEC 62769-5

Edition 3.0 2023-04
REDLINE VERSION

INTERNATIONAL STANDARD

colour inside

Field device integration (FDI®) –
Part 5: FDI Information Model

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.05 ISBN 978-2-8322-6828-5

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

FIELD DEVICE INTEGRATION (FDI®) –

Part 5: FDI® Information Model

FOREWORD

This redline version of the official IEC Standard allows the user to identify the changes made to the previous edition IEC 62769-5:2021. A vertical bar appears in the margin wherever a change has been made. Additions are in green text, deletions are in strikethrough red text.

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. It is an International Standard.

This third edition cancels and replaces the second edition published in 2021. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) added INTERACTIVE_TRANSFER_TO_DEVICE ACTION.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65E/858/CDV	65E/915/RVC

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

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 document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

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- withdrawn,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

IMPORTANT – The "colour inside" logo on the cover page of this document 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 IEC 62769 series has the general title *Field Device Integration (FDI)* and the following parts:~~

- ~~Part 1: Overview~~
- ~~Part 2: FDI Client~~
- ~~Part 3: FDI Server~~
- ~~Part 4: FDI Packages~~
- ~~Part 5: FDI Information Model~~
- ~~Part 6: FDI Technology Mapping~~
- ~~Part 7: FDI Communication Devices~~
- ~~Part 100: Profiles – Generic Protocol Extensions~~
- ~~Part 101-1: Profiles – Foundation Fieldbus H1~~
- ~~Part 101-2: Profiles – Foundation Fieldbus HSE~~
- ~~Part 103-1: Profiles – PROFIBUS~~
- ~~Part 103-4: Profiles – PROFINET~~
- ~~Part 109-1: Profiles – HART and WirelessHART~~
- ~~Part 115-2: Profiles – Protocol-specific Definitions for Modbus RTU~~
- ~~Part 150-1: Profiles – ISA 100.11a~~

FIELD DEVICE INTEGRATION (FDI®) –

Part 5: FDI® Information Model

1 Scope

This part of IEC 62769 defines the FDI®[1] 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 ~~a~~ 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.

[1] FDI® is a registered trademark of the non-profit organization Fieldbus Foundation, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

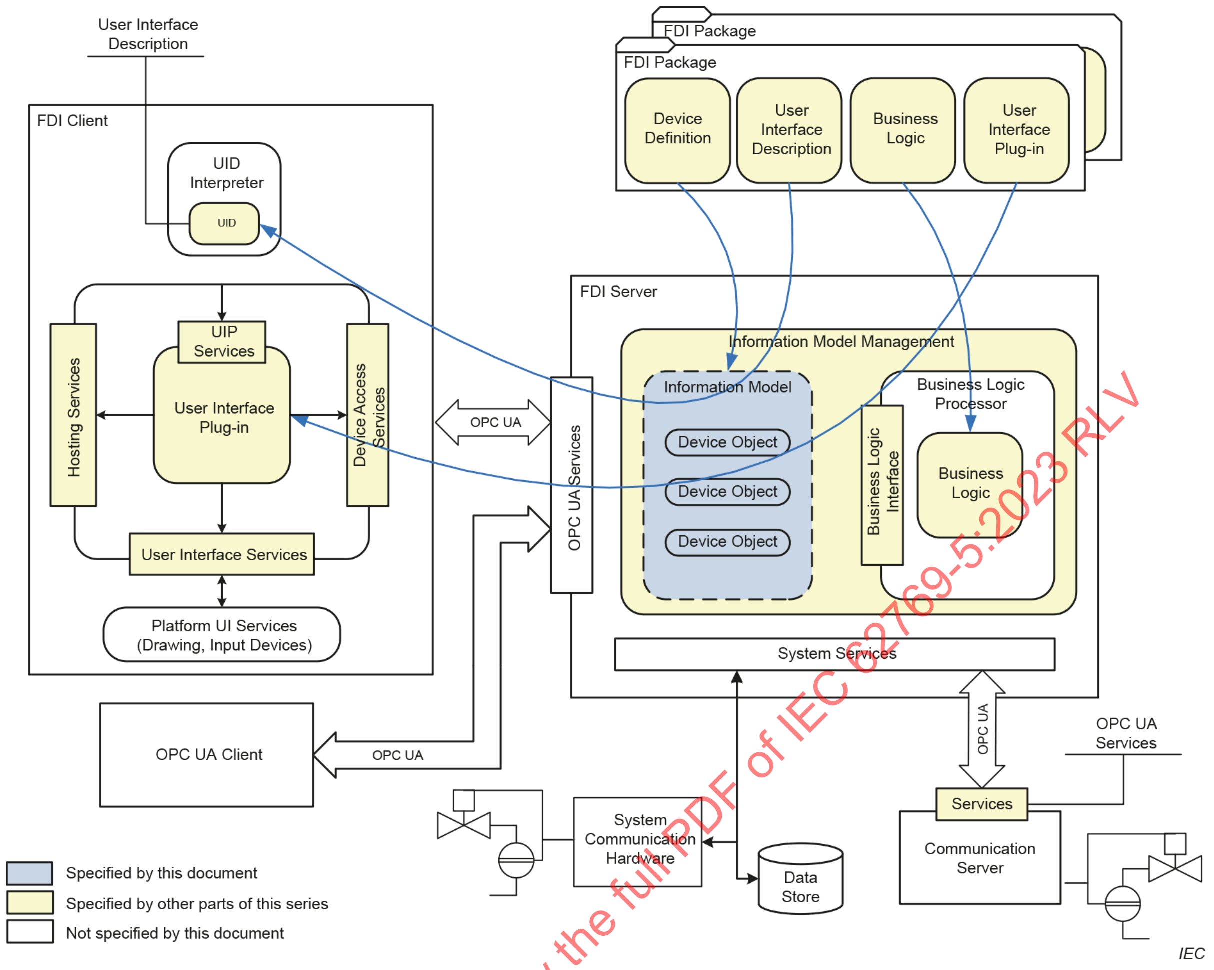


Figure 1 – FDI® architecture diagram

2 Normative references

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

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

IEC 61784-1-3:2023, *Industrial networks – Profiles – Part 1-3: Fieldbus profiles – Communication Profile Family 3*

IEC 61804-3, *Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 3: EDDL syntax and semantics*

IEC 61804-4, *Devices and integration in enterprise systems – Function blocks (FB) for process control and electronic device description language (EDDL) – Part 4: EDD interpretation*

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, *OPC Unified Architecture – Part 100: ~~OPC UA for Devices~~ Device Interface*

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

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

IEC 62769-3, *Field Device Integration (FDI®) – Part 3: Server*

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

IEC 62769-6, *Field Device Integration (FDI®) – Part 6: FDI® Technology Mappings*

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

IEC 62769-1xx (all parts), *Field Device Integration (FDI®) – Part 1xx-y: Profiles*

OPC 10000-19, *OPC Unified Architecture – Part 19: Dictionary Reference*

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 and IEC 62769-3 apply.

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

- IEC Electropedia: available at https://www.electropedia.org/
- ISO Online browsing platform: available at https://www.iso.org/obp

3.2 Abbreviated terms and acronyms

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

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

3.3 Conventions

~~For the purposes of this document, the conventions given in IEC 62769-1 apply.~~

3.3.1 Capitalization

Capitalization of the first letter of words is used in the IEC 62769 series to emphasize an FDI® defined term.

3.3.2 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 document. NodeClasses representing types always have a shadow.

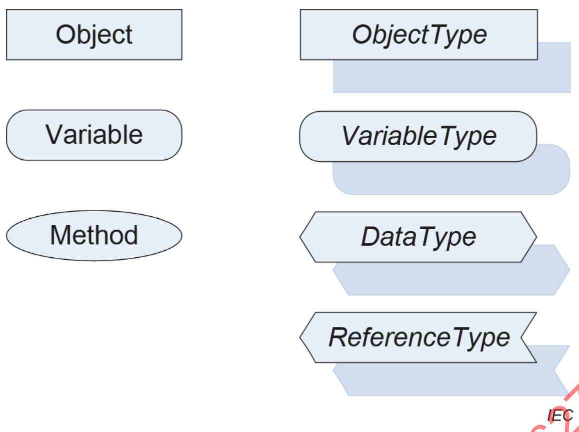


Figure 2 – OPC UA graphical notation for NodeClasses

Figure 3 shows the symbols for the ReferenceTypes used in this document. 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.

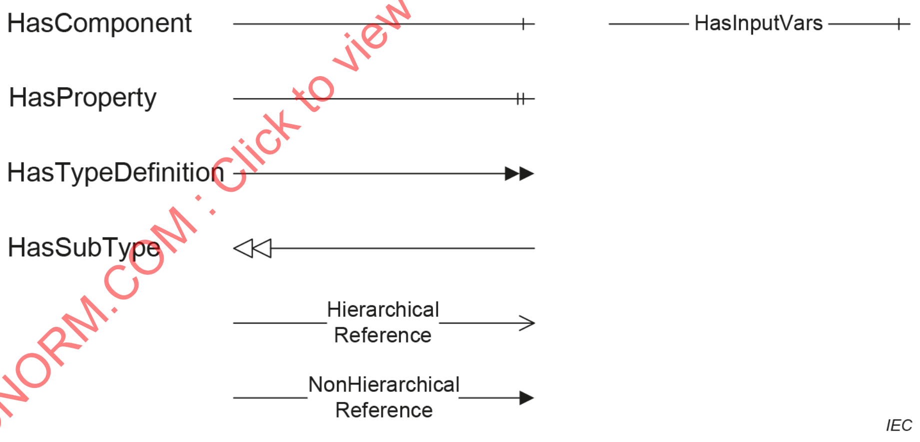


Figure 3 – OPC UA graphical notation for References

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).

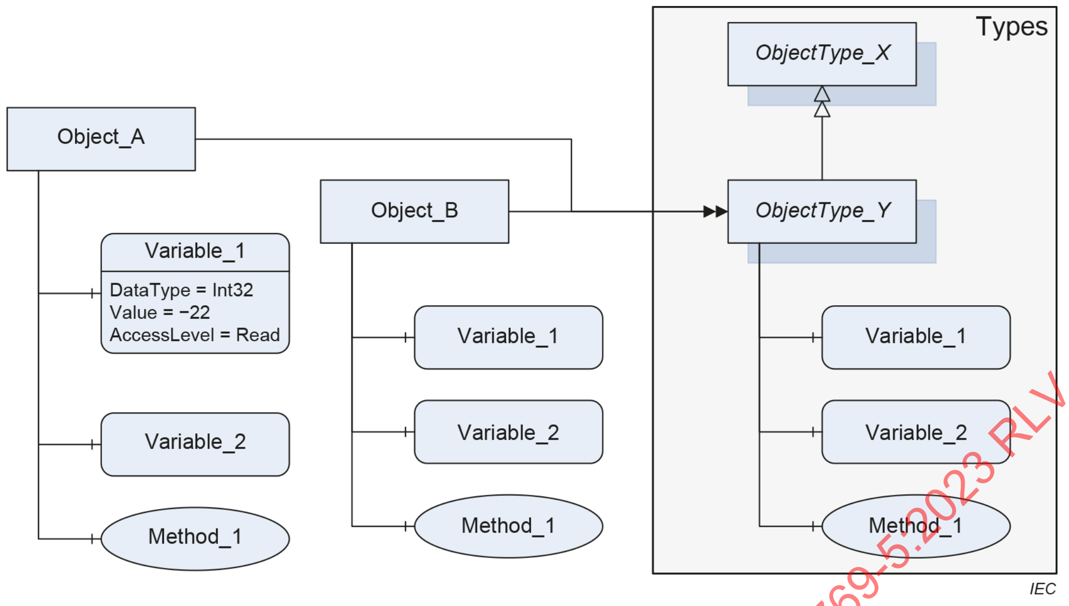


Figure 4 – OPC UA graphical notation example

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.

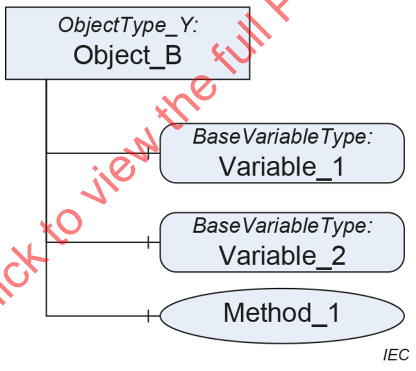


Figure 5 – Optimized Type Reference

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. ~~Numeric identifiers for NodeIds are defined in Annex A.~~

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~~ can have component DataVariables. This is typically used by Servers to expose individual elements of arrays and structures. This document 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. It is a DeviceType Object that is a subtype of TopologyElementType and inherits all components of this type. The component ParameterSet contains all Variables describing the Device. The component MethodSet contains all Methods provided by the Device. Components of the FunctionalGroupType are used to collect 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.

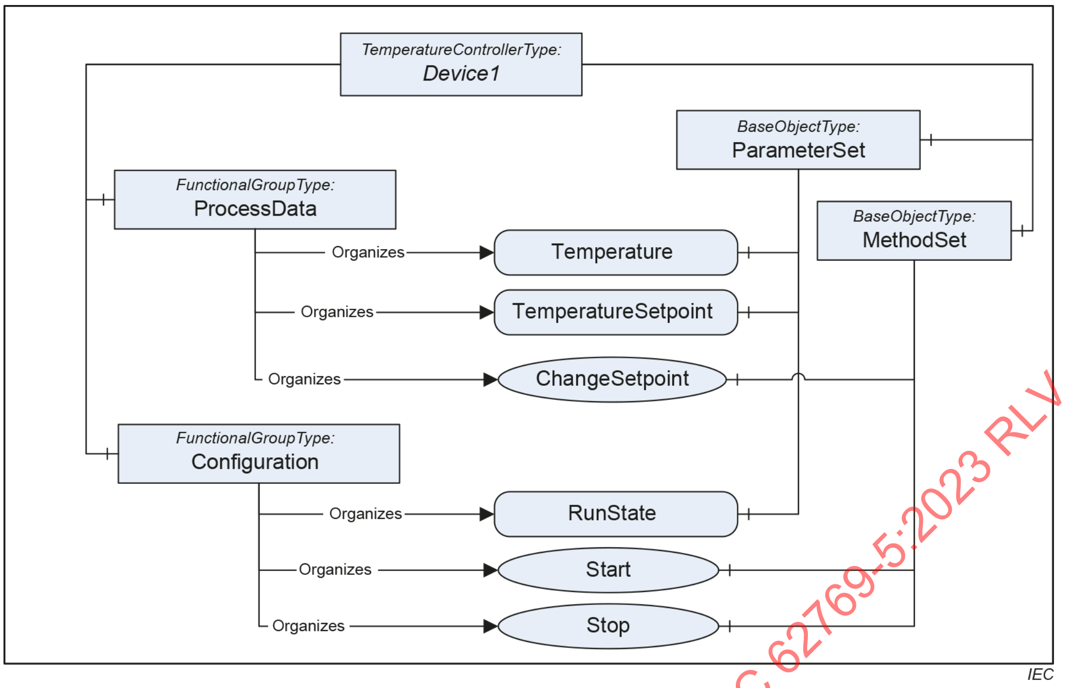


Figure 6 – OPC UA Devices example: Functional Groups

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.

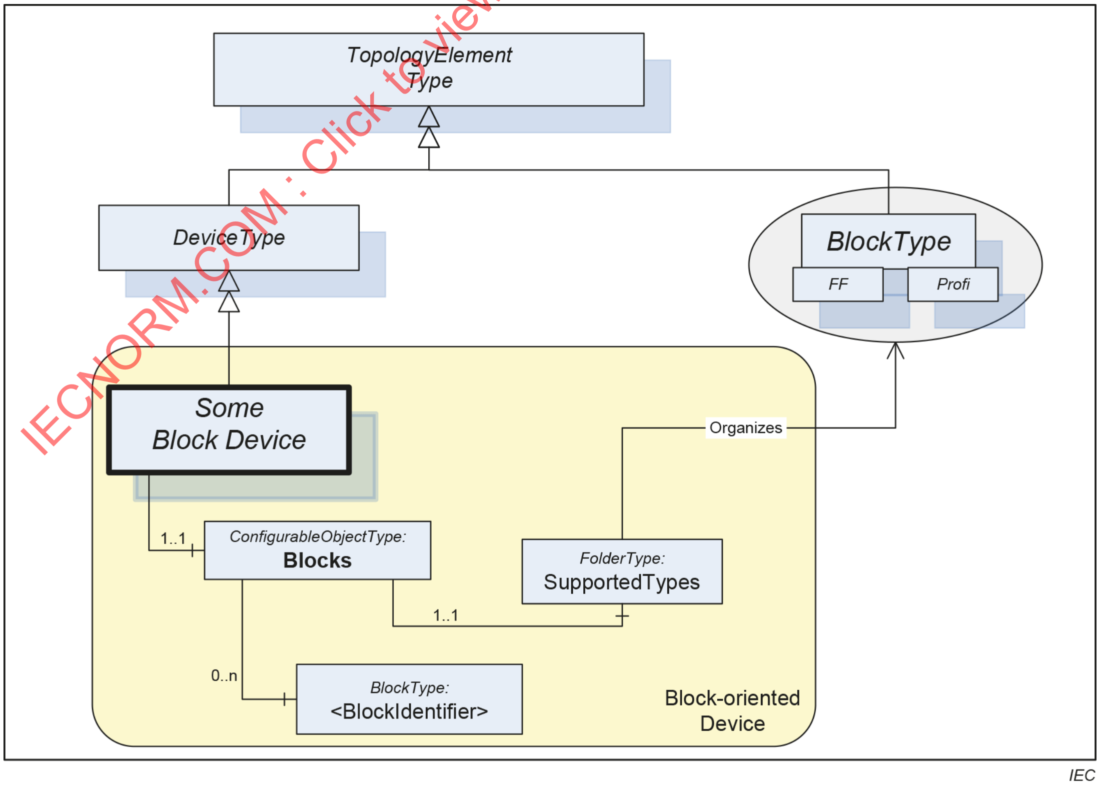


Figure 7 – OPC UA Devices example: Configurable components

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~~ can 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).

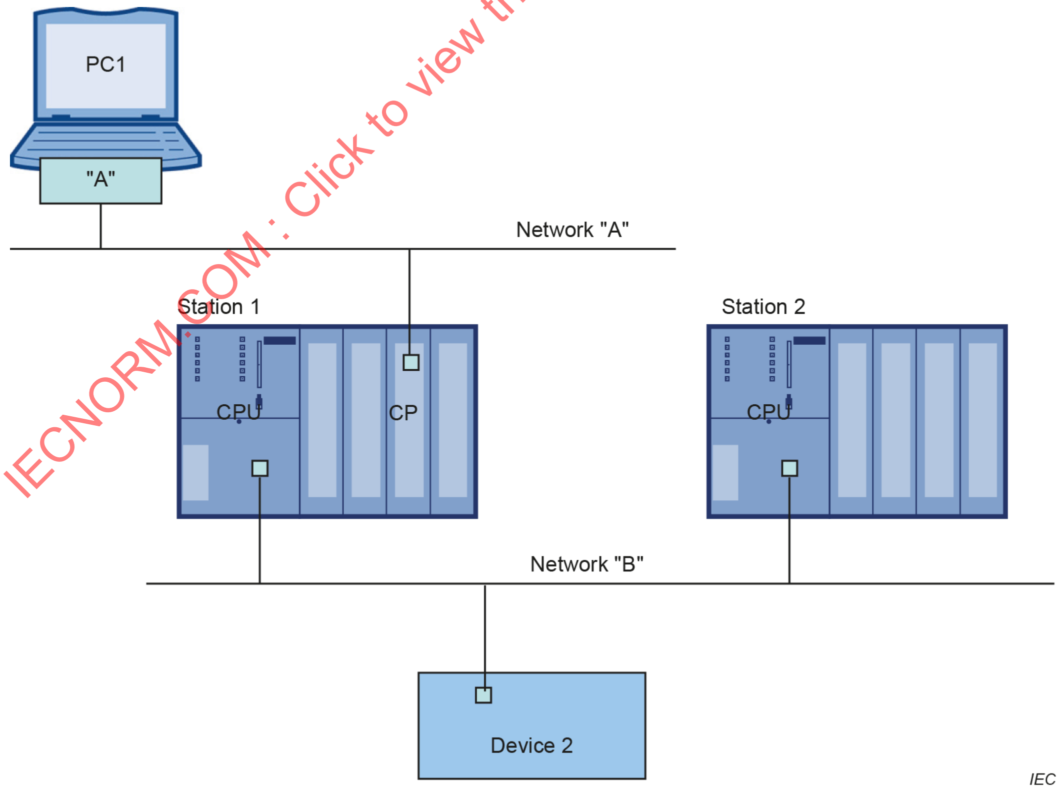


Figure 8 – Example of an automation system

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.

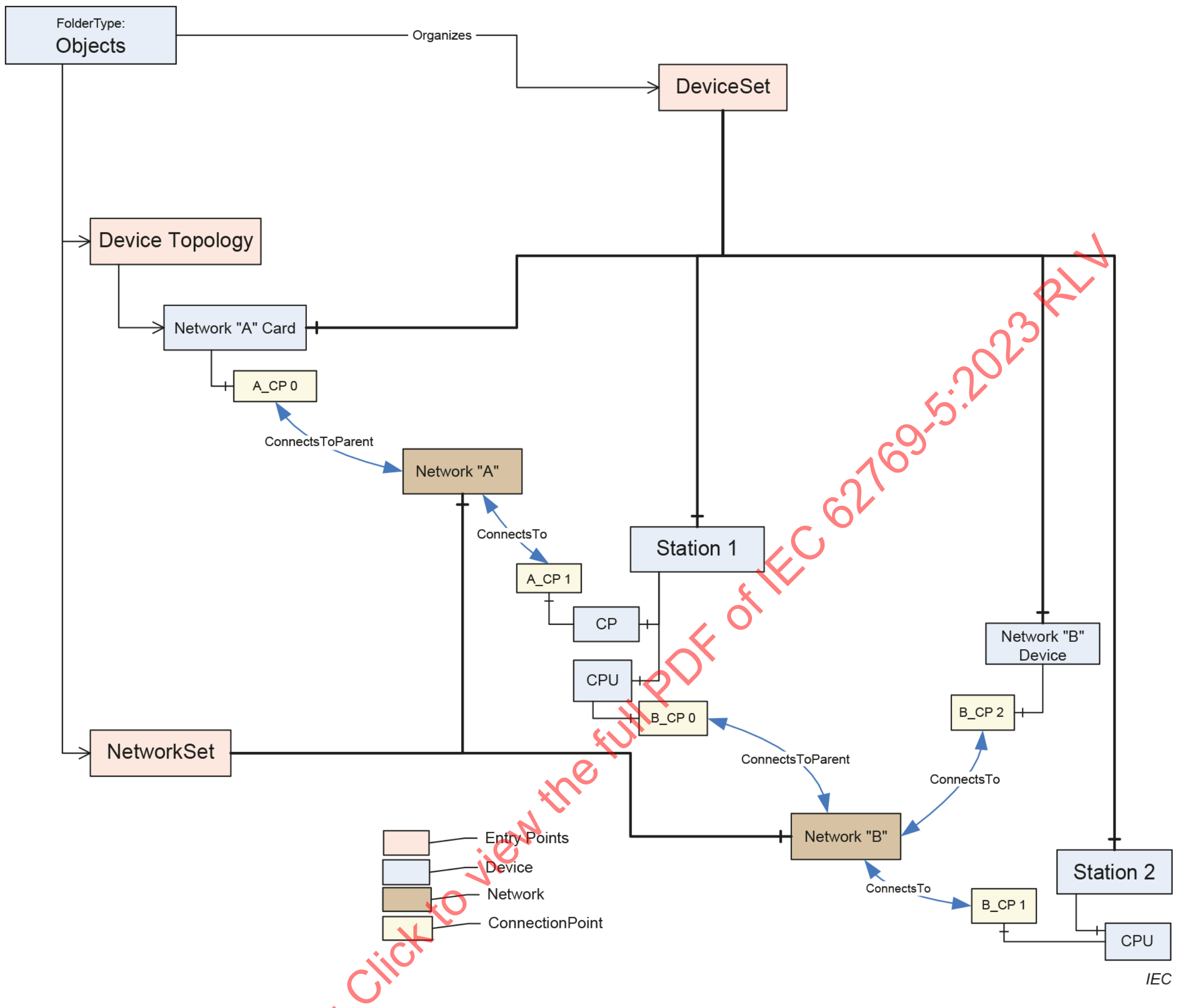


Figure 9 – Example of a Device topology

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 can exist in the AddressSpace.

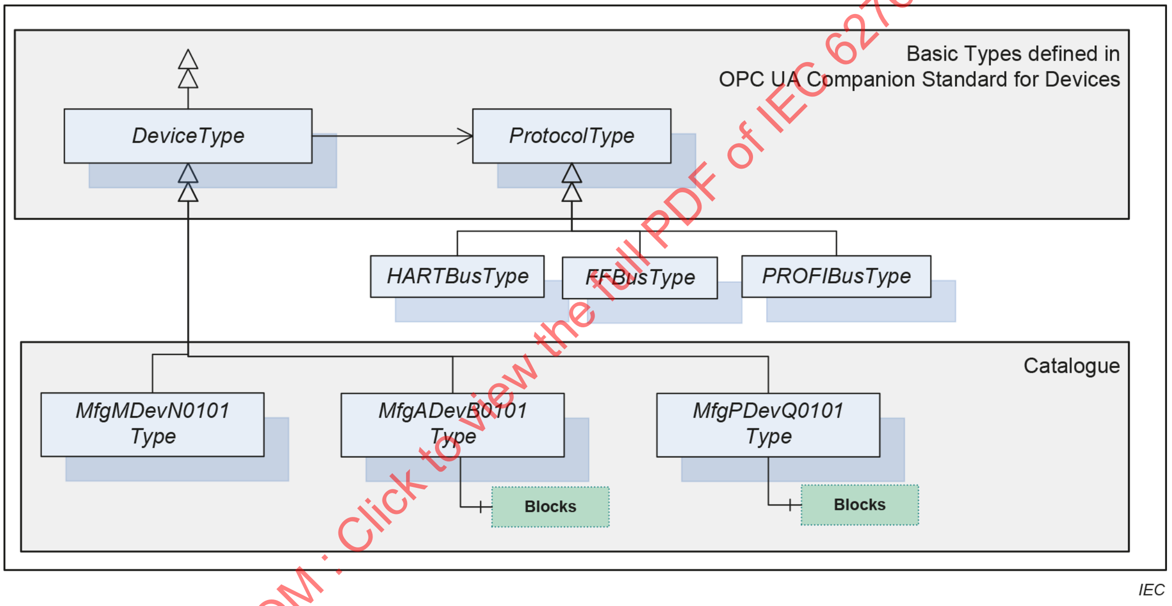


Figure 10 – Example Device Types representing a catalogue

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~~ shall go through Station 1, which provides the communication services for the PROFIBUS DP Network (see IEC 61784-1-3, CPF 3). This can be achieved through Nested Communication, which is specified in ~~IEC 62769-4~~ IEC 62769-3. Communication Devices and Communication Servers are specified in IEC 62769-7.

5.6 Semantic Information

The Type System of OPC UA already provides means to express the semantic of an Object. As an example, The Devices Companion Specification defines the DeviceType ObjectType expressing that instances of this ObjectType represent devices. However, the detailed semantic of the device is not specified further. Semantic information provides a means to represent that an Object has the semantic differential pressure transmitter for instance. This allows clients to automatically retrieve and identify such devices.

Objects in OPC UA typically have Variables holding variable values. While the BrowseName or DisplayName provide a "hint" regarding the semantic of the Variable, this cannot be used to automatically identify the semantic of a Variable. As an example, the ParameterSet of a DeviceType Object contains a flat list of the parameters of the device. Semantic information provides means to represent that a Variable (e.g. in the ParameterSet) has a specific semantic associated with it. This allows clients to automatically retrieve and identify such Variables.

The DictionaryEntryType Object, defined in OPC-10000-19, is used to add semantic information to objects in the FDI® Information Model. Subtypes of DictionaryEntryType shall define their own namespace (e.g. http://iec.ch/cdd/iec61987). The NodeId of Objects shall consist of the corresponding namespace index and the value of the ID Property as the identifier part. This allows direct access to the Semantic Object without further indirection. Annex A defines the numeric identifiers for all of the numeric NodeIds used in this document.

The OPC UA Server shall create a concrete object type derived from the abstract DictionaryEntryType for each dictionary referenced. Objects such as DeviceType, FunctionalGroupType, or VariableType can reference one or more DictionaryEntryType objects using the HasDictionaryEntry reference to indicate the semantic meaning for the object. The DictionaryId property of the DictionaryEntryType object contains the identifier value of an external dictionary entry. A client can use these references to access device data with only needing to know semantically what information is desired (i.e. specific object BrowseNames are not required).

The following example provides an overview of this design principle (Figure 11). This example uses the IEC Common Data Dictionary (CDD); however, it should be noted that any qualifying dictionary can be used. The IEC Common Data Dictionary (CDD) describes classes and properties with the purpose to characterize equipment. The IEC 61987 series for instance describes process automation equipment. Every class and property provide a set of attributes such as Version, Revision, Preferred name, Definition, etc. Most importantly, it describes a unique identifier for every class of property. In the case of IEC CDD this is an International Registration Data Identifier (IRDI). In this example, IEC61987DictionalEntryType is defined as a subtype of DictionaryEntryType. The Object DifferentialPressureTransmitter of this ObjectType contains the IEC CDD attribute values as its property values. The IRDI is set as the value of the ID property.

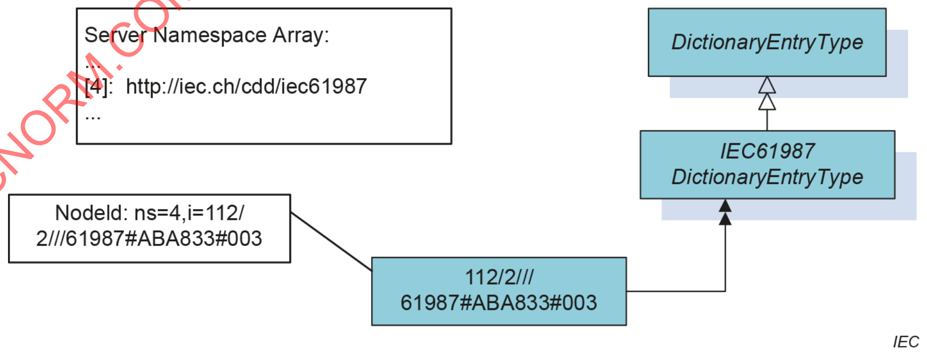

Server Namespace Array:
...
[4]: http://iec.ch/cdd/iec61987
...
DictionaryEntryType
IEC61987 DictionaryEntryType
NodeId: ns=4,i=112/2///61987#ABA833#003
112/2///61987#ABA833#003
IEC

Figure 11 – Example of concrete DictionaryEntryType and Object

Variables can be enumerations. In such cases, the Variable can have a specific semantic as well as each enumeration item of the Variable. As an example, a Variable can contain the units of the measurement value that can be configured. So, the semantic of the Variable itself is "measurement unit". The enumeration items of that Variable then stand for a specific unit of measure. In case of temperature, the first enumeration item can represent degrees Celsius, the second item represents degrees Fahrenheit, the third item represents degrees Kelvin and so on. Associating enumeration items with a specific semantic allows clients to automatically retrieve information such as the currently configured measurement unit without requiring interment knowledge of the particular unit code being used by the device.

Enumerated Variables contain an additional property DictionaryEntries, defined in 15.13, which is an array of DataType NodeId where each array item contains the NodeId of the corresponding DictionaryEntry Object. Figure 12 shows an example: The Variable Parameter2 has the Property DictionaryEntries. The array items of the DictionaryEntries Property contain the NodeIds of the corresponding DictionaryEntry Objects. In the example, Parameter2[0] is related to the Semantic1 Object, Parameter2[1] is not related to any Semantic Object and Parameter2[2] is related to the Semantic2 Object.

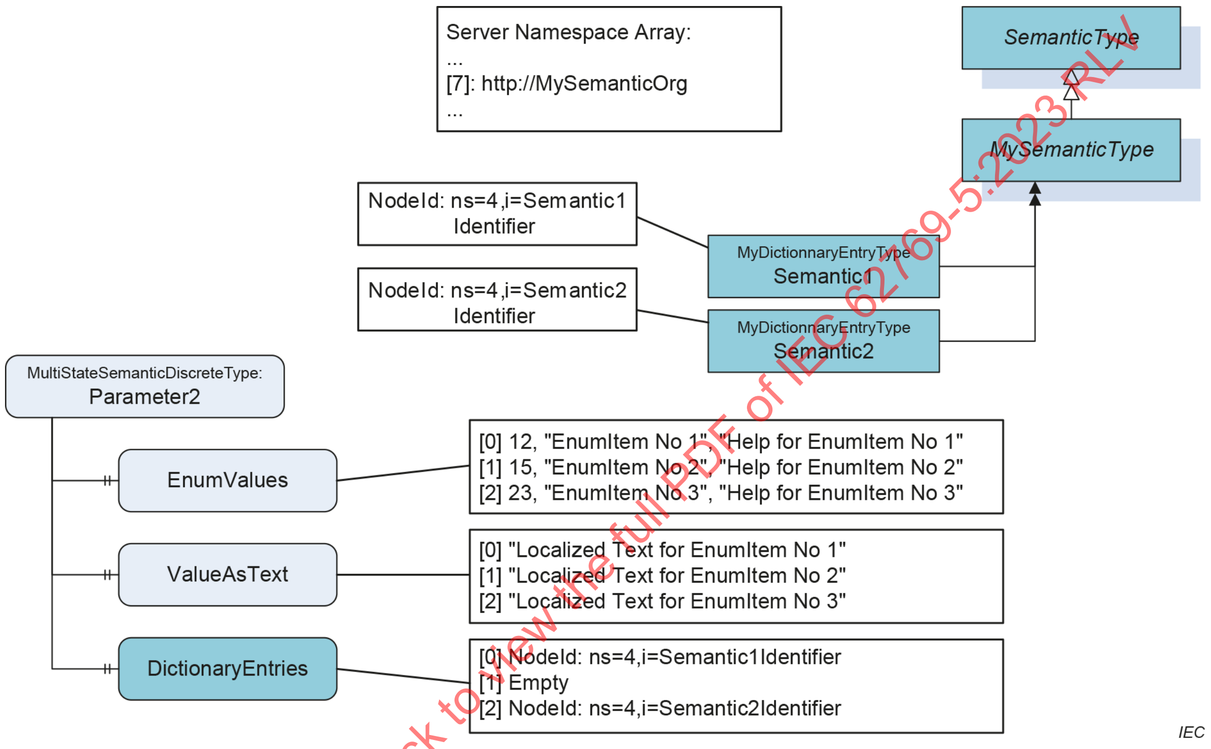

Server Namespace Array:
...
[7]: http://MySemanticOrg
...
SemanticType
MySemanticType
NodeId: ns=4,i=Semantic1 Identifier
NodeId: ns=4,i=Semantic2 Identifier
MyDictionnaryEntryType Semantic1
MyDictionnaryEntryType Semantic2
MultiStateSemanticDiscreteType: Parameter2
EnumValues
ValueAsText
DictionaryEntries
[0] 12, "EnumItem No 1", "Help for EnumItem No 1"
[1] 15, "EnumItem No 2", "Help for EnumItem No 2"
[2] 23, "EnumItem No 3", "Help for EnumItem No 3"
[0] "Localized Text for EnumItem No 1"
[1] "Localized Text for EnumItem No 2"
[2] "Localized Text for EnumItem No 3"
[0] NodeId: ns=4,i=Semantic1Identifier
[1] Empty
[2] NodeId: ns=4,i=Semantic2Identifier
IEC

Figure 12 – Example of DictionaryEntries

The custom query method GetNodeIdByDictionaryEntryId, defined in 15.14, can be used by a client to search for the nodes that are referencing a specified dictionary entry id. This provides an efficient means for finding data using semantic information.

6 AddressSpace organization

To promote interoperability of FDI® Clients and FDI® Servers, a set of Objects and relationships are defined in ~~Clauses 7, 8, 9, and 10~~ the following Clause 7. 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:

- The Types-part contains information about all components that have been generated based on descriptive information from FDI® Packages (see 5.4).
- 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~~ can 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 document) 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 13.

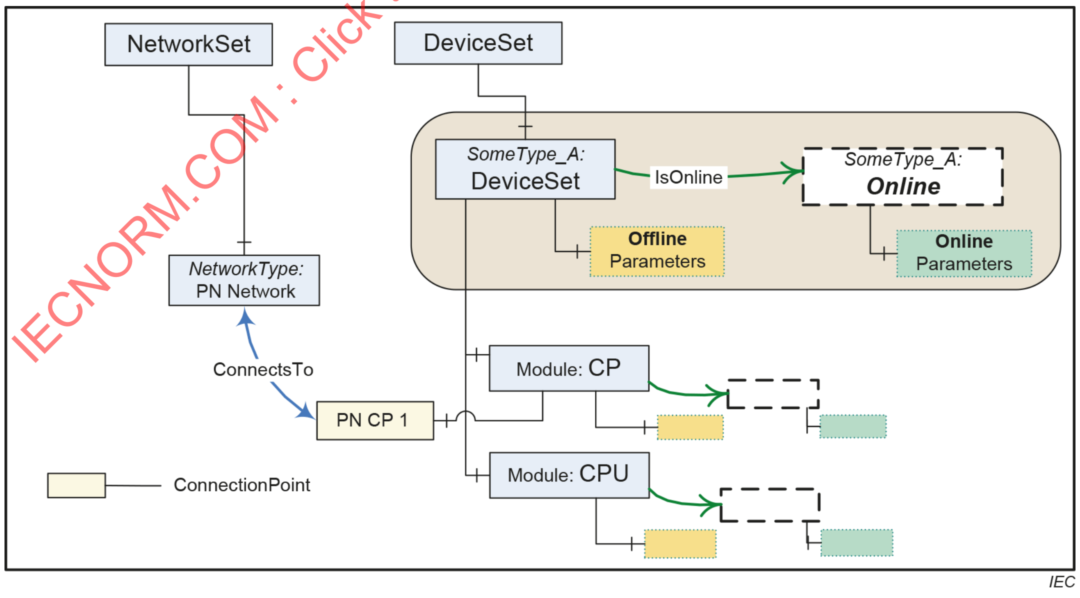


Figure 13 – Online component for access to device data

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

7.3.1 DeviceHealth Mapping

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~~ can vary from several seconds up to minutes.

The mapping of the GetHealthStatus return values to OPC UA is specified in Table 1.

Table 1 – DeviceHealth Mapping

GetHealthStatus	OPC UA
0 – Indeterminate	Indeterminate is not defined in the the DeviceHealth data type in IEC 62541-100. Servers that cannot determine the health status of the device shall return the OPC UA Read operation for this Property with an appropriate OPC UA status code, for example: Bad_NotConnected Bad_OutOfService Bad_NoCommunication Bad_NotSupported
The following values can be mapped to corresponding values defined for the DeviceHealth data type in IEC 62541-100.	
GetHealthStatus	**DeviceHealth data type values**
1 – Failure	FAILURE_1
2 – Function Check	CHECK_FUNCTION_2
3 – Out of Specifications	OFF_SPEC_3
4 – Maintenance Required	MAINTENANCE_REQUIRED_4
5 – Good	GOOD_0

Different to OPC UA for Devices (IEC 62541-100), support of the DeviceHealth Property is mandatory for Device Objects in FDI®. This is illustrated in Table 2. The complete DeviceType model and its formal definition are in IEC 62541-100.

Table 2 – DeviceType definition (excerpt applicable for Subclause 7.3.1)

Attribute	Value				
BrowseName	DeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the TopologyElementType defined in IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	~~Optional~~ → Mandatory
………					

7.3.2 DeviceHealth Diagnostics

In certain cases, a Device ~~may~~ can have additional information associated with the health status, e.g. the possible cause of an abnormal DeviceHealth status and suggested actions to return to normal.

This additional information is available with the DeviceHealthDiagnostics Variable. It is formally defined in Table 3.

Table 3 – DeviceType definition with DeviceHealth and DeviceHealthDiagnostics

Attribute	**Value**				
BrowseName	DeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the TopologyElementType defined in IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	Mandatory
HasComponent	Variable	DeviceHealthDiagnostics[]	LocalizedText	BaseDataVariableType	Mandatory
………					

DeviceHealthDiagnostics is an array of LocalizedText. Each array element contains related diagnostic information. The language shall match the requested locale specified in the OPC UA Session. See 15.2.1 on how to select the proper language for the current OPC UA Session.

When DeviceHealth is requested, the Server shall always also fetch the matching DeviceHealthDiagnostics. The Server shall assure that the values are synchronized by executing GetHealthStatus first and then reading the DeviceHealthDiagnostics. The Server will cache the DeviceHealthDiagnostics as part of the OPC UA Session until DeviceHealth is requested again.

If DeviceHealthDiagnostics is read with a separate service request, the Server shall return the diagnostic information associated with the most recently read DeviceHealth status. If the DeviceHealth status has not been read in the current OPC UA Session, the Server shall return Bad_NotReadable.

If no diagnostic information is available, the returned Value is Null.

See IEC 62769-4 on how DeviceHealthDiagnostics maps to the corresponding EDD information.

Example of a DeviceHealthInformation value with two array elements:

- [0]:
  "Critical Power Failure\n
  Possible cause: Critical Power failure has occurred\n
  Suggested action: Please check device/surroundings/process.\n"
- [1]:
  "Device Malfunction\n
  Possible cause: The device has detected a serious hardware error or failure.\n
  Suggested action: Check detailed device diagnosis if possible and/or replace device hardware if necessary.\n"

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 14 illustrates the type hierarchy of the user interface elements defined in IEC 62541-100 and in this document.

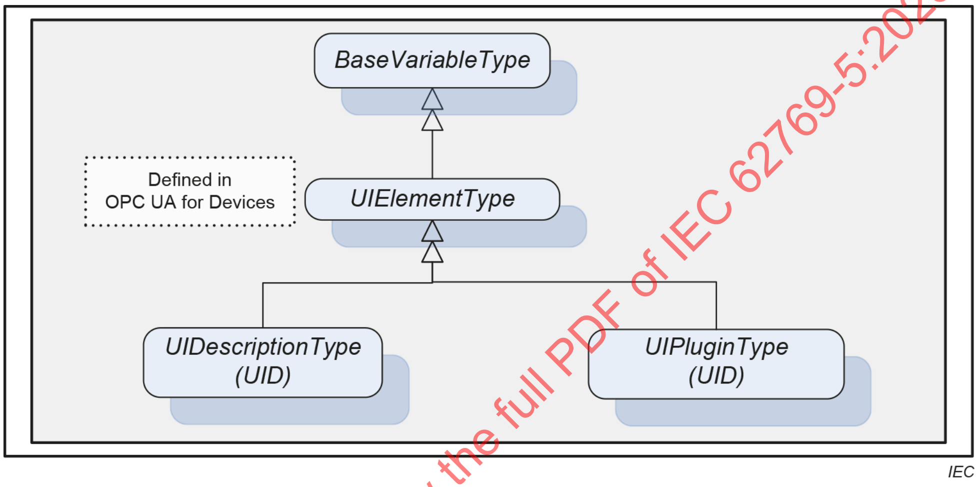


Figure 14 – Hierarchy of user interface Types

7.4.2 UI Description Type

FDI® Servers ~~may~~ can 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 4.

Table 4 – 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 5.

Table 5 – 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~~ can 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~~ might 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~~ can be large. Reading large values with a single Read operation ~~may~~ might 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~~ can 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~~ can 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 15 illustrates how Actions are integrated into a TopologyElement.

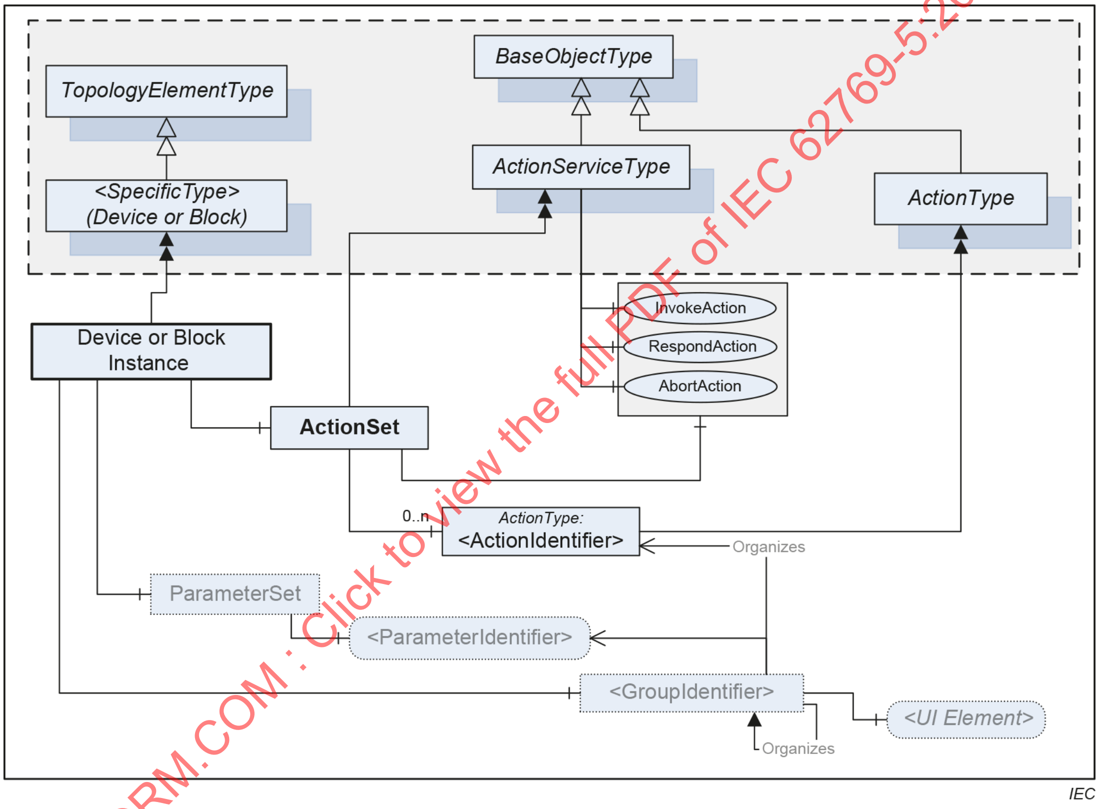


Figure 15 – Integration of Actions within a TopologyElement

ActionSet is used to aggregate the Actions for a specific TopologyElement. This Object is not available on the Type and is only available in an instance if Actions exist for this TopologyElement.

Actions can also be referenced from FunctionalGroup Objects.

7.6.2 Action Type

This ObjectType defines the structure of an Action. It is formally defined in Table 6.

Table 6 – 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 7. Its use is illustrated in Figure 15.

Table 7 – 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 document. 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 16.

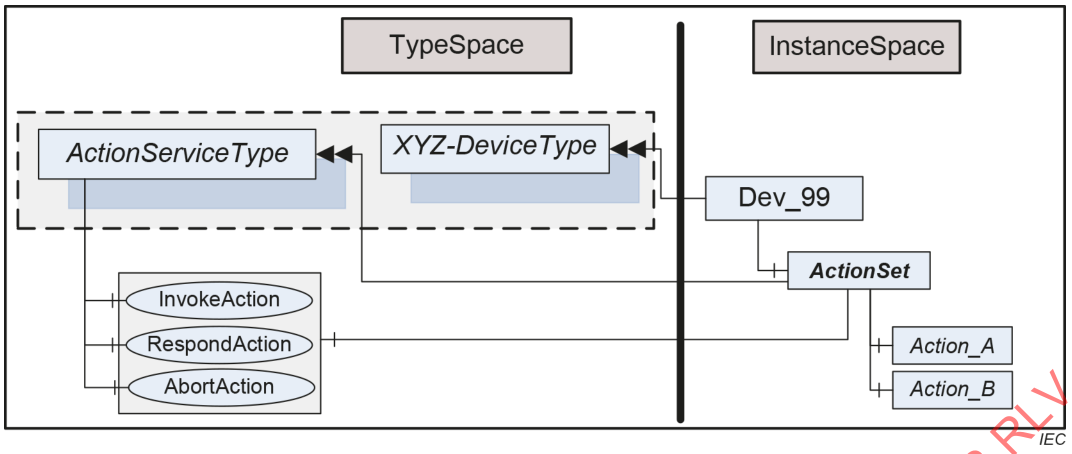


Figure 16 – Action Service

This Object is used as container for the ActionService Methods and shall have the BrowseName ActionSet. It is formally defined in Table 7. 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 has ~~had~~ a chance to subscribe.

The signature of this Method is specified below. Table 8 and Table 9 specify the arguments and AddressSpace representation, respectively.

Signature

```
    InvokeAction(
      [in]  String       ActionName,
      [in]  String       MethodArguments,
      [out] NodeId       ActionNodeId,
      [out] Int32        InvokeActionError);
```

Table 8 – 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 9 – 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 10 and Table 11 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 10 – 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 11 – 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 12 and Table 13 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 12 – 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 13 – 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

7.6.8 Interactive Transfer to device

The functionality to transfer data to the online Device including user interaction is provided as INTERACTIVE_TRANSFER_TO_DEVICE ACTION by the FDI® Server. This action has no arguments. It is started using the InvokeAction method.

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 62769-3. 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~~ can 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 14.

Table 14 – 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 17.

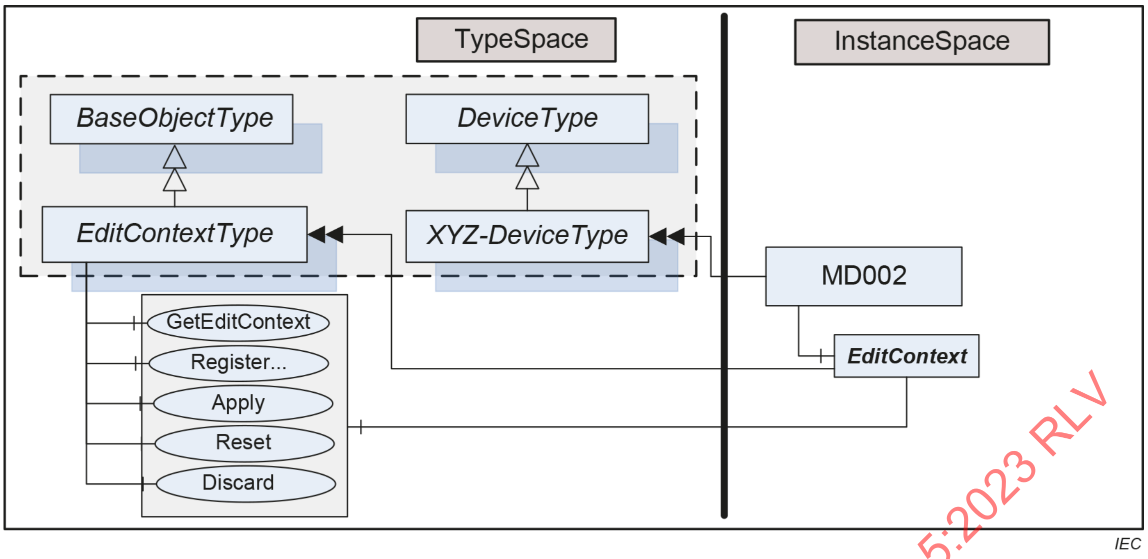


Figure 17 – EditContext type and instance

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~~ can 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 15 and Table 16 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 15 – 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 16 – 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 17 and Table 18 specify the arguments and AddressSpace representation, respectively.

Signatures

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

Table 17 – 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 18 – 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 19 – 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 20.

Table 20 – 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 offline representation of the Node in the EditContext. See IEC 62541-3 for details.
offlineDeviceNodeId	NodeId	This NodeId shall be used to address the offline 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 21 and Table 22 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 21 – 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 22 – 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 23.

Table 23 – 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 24 and Table 25 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 24 – 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 25 – 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 26 and Table 27 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 26 – 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 27 – 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 DirectDeviceAccess

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~~ can 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 18 shows the DirectDeviceAccessType definition. It is formally defined in Table 28.

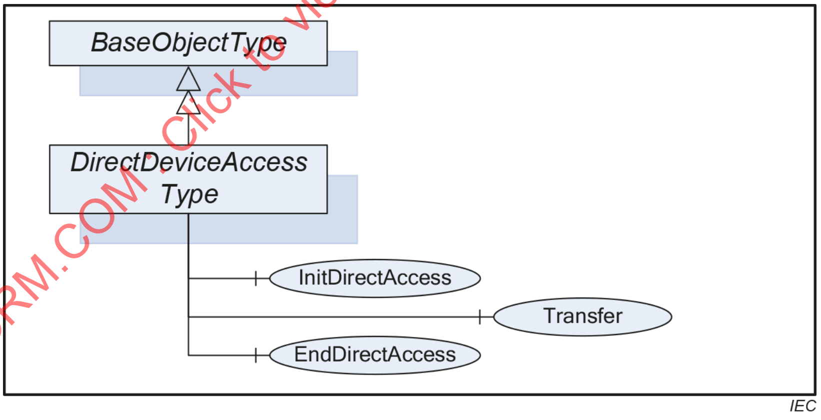


Figure 18 – DirectDeviceAccessType

Table 28 – DirectDeviceAccessType Definition

Attribute	Value				
BrowseName	~~DirectDeviceAccessType~~ Discard				
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 document. 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 19.

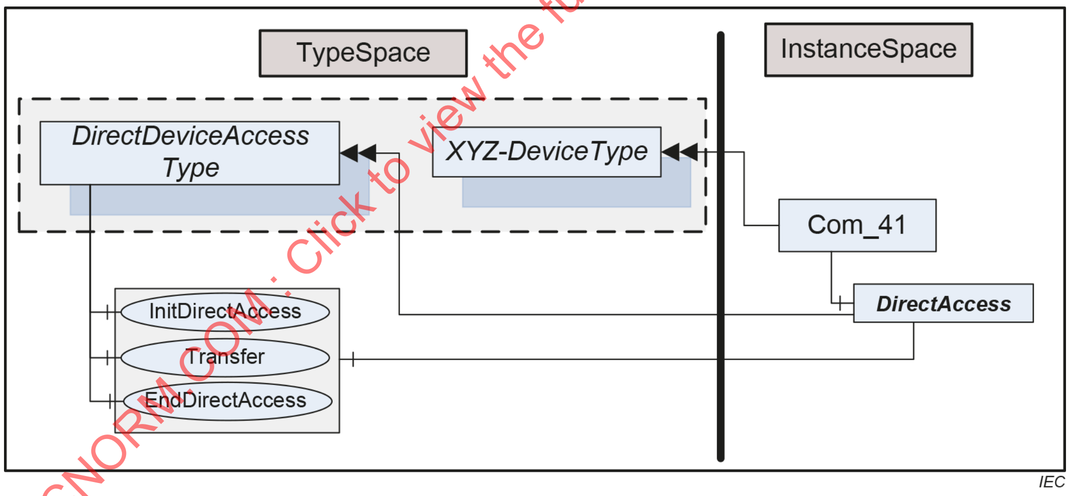


Figure 19 – DirectDeviceAccess instance

This Object is used as container for the DirectDeviceAccess Methods and shall have the BrowseName DirectAccess. It is formally defined in Table 29. 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 29 – 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 30 and Table 31 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 30 – 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 cannot be directly accessed -2 – E_LockRequired – the element is not locked as required -3 – E_InvalidState – the device is already in DirectAccess mode

Table 31 – 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 32 and Table 33 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 32 – 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 33 – 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 34 and Table 35 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 34 – 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 and IEC 62769-1xx series.
ReceiveData	XML document based on the TransferResultDataType as specified in the communication profile-specific XML schema. See IEC 62769-4 and IEC 62769-1xx series.
TransferError	0 – OK -1 – E_InvalidState – the device is not in DirectAccess mode.

Table 35 – 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 behaviours 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 20. 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.

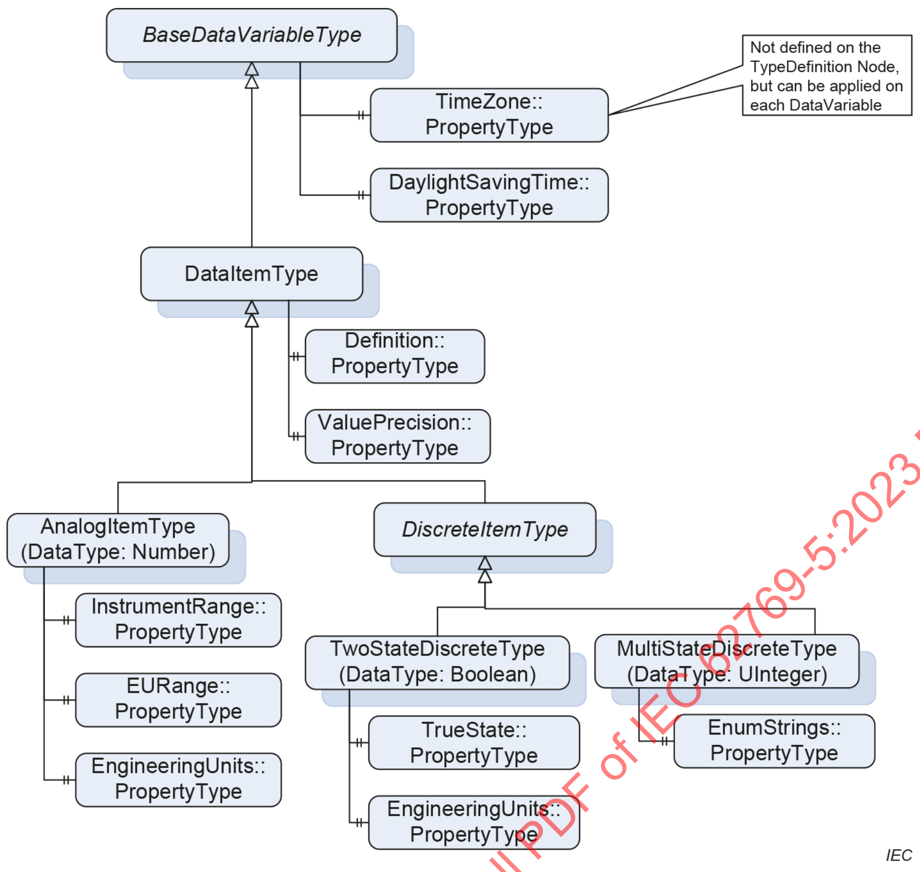

BaseDataVariableType
TimeZone:: PropertyType
DaylightSavingTime:: PropertyType
Not defined on the TypeDefinition Node, but can be applied on each DataVariable
DataItemType
Definition:: PropertyType
ValuePrecision:: PropertyType
AnalogItemType (DataType: Number)
InstrumentRange:: PropertyType
EURange:: PropertyType
EngineeringUnits:: PropertyType
DiscreteItemType
TwoStateDiscreteType (DataType: Boolean)
TrueState:: PropertyType
EngineeringUnits:: PropertyType
MultiStateDiscreteType (DataType: UInteger)
EnumStrings:: PropertyType
IEC

Figure 20 – OPC UA VariableTypes including OPC UA DataAccess

The mapping of EDD Data Types to OPC UA DataTypes and VariableTypes is specified in 15.6.

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 36.

Table 36 – 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 37.

Table 37 – 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 38.

Table 38 – 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 39.

Table 39 – Variant_Range Definition

Attributes	Value
BrowseName	Variant_Range

11 FDI® StatusCodes

11.1 General

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

11.2 Structure of the StatusCode

The general structure of the StatusCode is specified in IEC 62541-4. The exact bit assignments based on IEC 62541-4 are shown in Table 40.

Table 40 – StatusCode Bit Assignments

Field	Bit Range	Description
Severity	30 .. 31	Indicates whether the StatusCode represents a good, bad or uncertain condition. These bits have the following meanings:

Severity	Bits	Description
Good Success	00	The operation was successful; results may be used.
Uncertain Warning	01	The operation was partially successful; results might not be suitable for some purposes.
Bad Failure	10	The operation failed and any associated results cannot be used.
Reserved	11	Reserved for future use. Should also be treated as "Bad".

Field	Bit Range	Description
Reserved	29 .. 28	Reserved for future use. Shall always be zero.
SubCode	16 .. 27	The code is a numeric value assigned to represent different conditions. Each code has a symbolic name and a numeric value. All descriptions in this specification refer to the symbolic name. The numeric values are defined within the FDI® Type Libraries..
Reserved	12 .. 15	Reserved for future use. Shall always be zero.
InfoType	10 .. 11	The type of information contained in the info bits. These bits have the following meanings:

Severity	Bits	Description
NotUsed	00	The info bits are not used and shall be set to zero.
DataValue	01	The StatusCode and its info bits are associated with a data value returned from the FDI® Server.
Reserved	1X	Reserved for future use. The info bits shall be ignored.

Field	Bit Range	Description
InfoBits	0 .. 9	Additional information bits that depend on the Info Type field.

Table 41 describes the structure of the InfoBits when the Info Type is set to DataValue (01).

Table 41 – DataValue InfoBits

Info Type	Bit Range	Description
LimitBits	8 .. 9	The limit bits associated with the data value. The limits bits have the following meanings:

Limit	Bits	Description
None	00	The value is free to change.
Low	01	The value is at the lower limit for the data source.
High	10	The value is at the higher limit for the data source.
Constant	11	The value is constant and cannot change.

Info Type	Bit Range	Description
Overflow	7	If this bit is set, not every detected change has been returned since the FDI® Server's queue buffer for the subscribed Variable reached its limit and had to purge out data.
Reserved	0 .. 6	Reserved for future use. Shall always be zero.

11.3 FDI® specific operation level result codes

IEC 62541-4 includes a set of common operational result codes which also apply to FDI®. Table 42 contains Good (success) codes that are specifically defined for FDI®.

Table 42 – 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~~ all the following conditions apply: – the value of the Variable has not been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Good_Edited _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 all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Good_Edited _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Good_Edited _DominantValueChanged _DependentValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of its dominant Variable was changed; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.

Table 43 contains Uncertain codes that are specifically defined for FDI®.

Table 43 – Uncertain operation level result codes

Symbolic Id	Description
Uncertain_DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when a dominant value – e.g. an engineering unit – was changed and the dependent Variable ~~may~~ might have to be recalculated.
Uncertain_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. The "uncertain" status indicates that the dominant variable value itself has uncertain quality.
Uncertain _NoCommunicationLastUsableValue	Communication to the data source has failed. The Variable value is the last value that had a good quality.
Uncertain_LastUsableValue	Whatever was updating this value will no longer be doing so.
Uncertain_SubstituteValue	The value is an operational value that was manually overwritten.
Uncertain_InitialValue	The value is an initial value for a Variable that normally receives its value from another Variable.
Uncertain_SensorNotAccurate	The value is at one of the sensor limits.
Uncertain_EngineeringUnitsExceeded	The value is outside of the range of values defined for this parameter.
Uncertain_SubNormal	The value is derived from multiple sources and has less than the required number of good sources.

Table 44 contains Bad codes that are specifically defined for FDI®.

Table 44 – Bad operation level result codes

Symbolic Id	Description
Bad_Edited_OutOfRange	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value).
Bad_InitialValue_OutOfRange	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a Variable value when all the following conditions apply: – the value is an initial value for a Variable that normally receives its value from another Variable; – the value is not a legal value (e.g. out of range or an invalid enumeration value).
Bad_DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when a dominant value – e.g. an engineering unit – was changed and the dependent Variable cannot be accessed.
Bad_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. ~~The "uncertain" status indicates that the dominant variable value itself has bad quality.~~
Bad_OutOfRange _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.

Symbolic Id	Description
Bad_Edited _OutOfRange _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Bad_OutOfRange _DominantValueChanged _DependenValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Bad_Edited _OutOfRange _DominantValueChanged _DependenValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Bad_ConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
Bad_DeviceFailure	There has been a failure in the device/data source that generates the value.
Bad_NodeInvalid	The identifier does not refer to a valid Node in the Device Model. This result code is used both as service- and as operation-level result code.
Bad_NotConnected	The Variable should receive its value from another Variable, but has never been configured to do so.
Bad_OutOfService	The source of the data is not operational.
Bad_SensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source.
Bad_UIPHandleInvalid	The handle does not refer to a subscribed Node Attribute.
Bad_WaitingForInitialData	Waiting for the FDI® Server to obtain values from the underlying data source. After subscribing to Variables, it might take some time before values are delivered. In such cases, an initial update might be sent with this status prior to the Notification with the first valid value.

12 Specialized topology elements

Devices and other topology elements can be specialized using the modelling elements defined in this document 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 45 and Table 46 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 45 – LogAuditTrailMessage Method Arguments

Argument	Description
Message	Free text describing the context.

Table 46 – 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 47.

Table 47 – 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~~ can 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~~ can 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 48 specifies the mapping of Properties to Package Catalog elements.

Table 48 – 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 15.6 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 15.8 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 Modular Device

Subclause 15.4 specifies the mapping of a modular device's package.

As described in IEC 62769-4, a package for a modular device contains a head station EDD and one or more module EDDs. The head station EDD contains COMPONENT constructs that identify the module EDDs.

The EDDs for head station Device and for each module shall be individually mapped according to the mapping rules for single Devices.

The head station Device will show up as a modular Device as specified in IEC 62541-100.

- The SubDevices component in the instance of the head Device references instantiated modules.
- The SupportedTypes folder in the SubDevices component references all DeviceTypes for modules that ~~may~~ can appear in the instance of the head Device.

15.5 Block

15.5.1 General

Subclause 15.5 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~~ can appear in a Device Instance. The Blocks component in a Device Instance references instantiated Blocks.

15.5.2 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.

15.5.3 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 exists on the Type, the offline and the online instances.

See 15.6 on how EDDL attributes are used to create Parameter nodes.

15.5.4 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.

15.5.5 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 15.8 on how EDDL attributes are used to create Action Nodes.

15.5.6 Mapping to MethodSet

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

15.5.7 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.

15.6 Parameter

15.6.1 General

EDDL Parameters (for devices or blocks) are mapped to OPC UA Variables. The VariableType used ~~may~~ can 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 document includes additional VariableTypes for more sophisticated EDDL types. See Table 50 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 49 summarizes the Variable Attributes and describes how they are set from the EDDL description information.

Table 49 – Setting OPC UA Variable Attributes from EDDL variable attributes

Attributes	Description
Value	The most recent value of the Variable that the FDI® Server has read from the device.
DataType	The EDDL data type is translated into an OPC UA standard data type according to Table 50.
ValueRank	Either set to "Scalar" or – when the Parameter is an array – the number of dimensions.
ArrayDimensions	Specifies the length of each dimension if the Parameter is an array. This attribute is not exposed if the length is unknown or dynamic.
AccessLevel	The AccessLevel Attribute is set based on the EDDL variable HANDLING attribute according to the following table:

Field	Bit	Description
CurrentRead	0	Set if EDDL variable HANDLING is defined as READ. Reset otherwise.
CurrentWrite	1	Set if EDDL variable HANDLING is defined as WRITE. Reset otherwise.

Attributes	Description
	If the HANDLING attribute is missing, the Parameter will be defined as readable and writeable.
UserAccessLevel	The AccessLevel with possible restrictions based on client identity as defined by the FDI® Server.
MinimumSamplingInterval	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).

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 50 shows the correspondence between the EDDL types and sizes and the OPC UA VariableTypes and standard DataTypes.

Table 50 – 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 15.6.5	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 15.6.6	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 intervals since January 1, 1601 (UTC))	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 service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
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	TIME_VALUE is the "number of 1/32 ms". The "Duration" is calculated by multiplying the TIME_VALUE with 0,031 25 (which is the float equivalent for 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.

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
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 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 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.
OCTET	BaseDataVariableType	ByteString	
INDEX	See 15.6.5	String	
BOOLEAN	BaseDataVariableType	Boolean	

As Table 50 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. Subclause 15.6 is intended to provide details for each EDDL data type.

15.6.2 Private Parameters

The Parameters specified in an FDI® Package ~~may~~ can be declared private using the PRIVATE Attribute specified in IEC 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.

15.6.3 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.

15.6.4 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 affected variables using that unit code shall be refreshed. When an affected 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.

15.6.5 Enumerated Parameters

If no semantic map has been applied to an enumerated variable, an OPC UA DataVariable with the VariableType MultiStateValueDiscreteType (defined in IEC 62541-8) is used for each enumerator in the EDDL enumerated variable definition – otherwise the VariableType MultiStateDictionaryEntryDiscreteType shall be used.

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.

15.6.6 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.

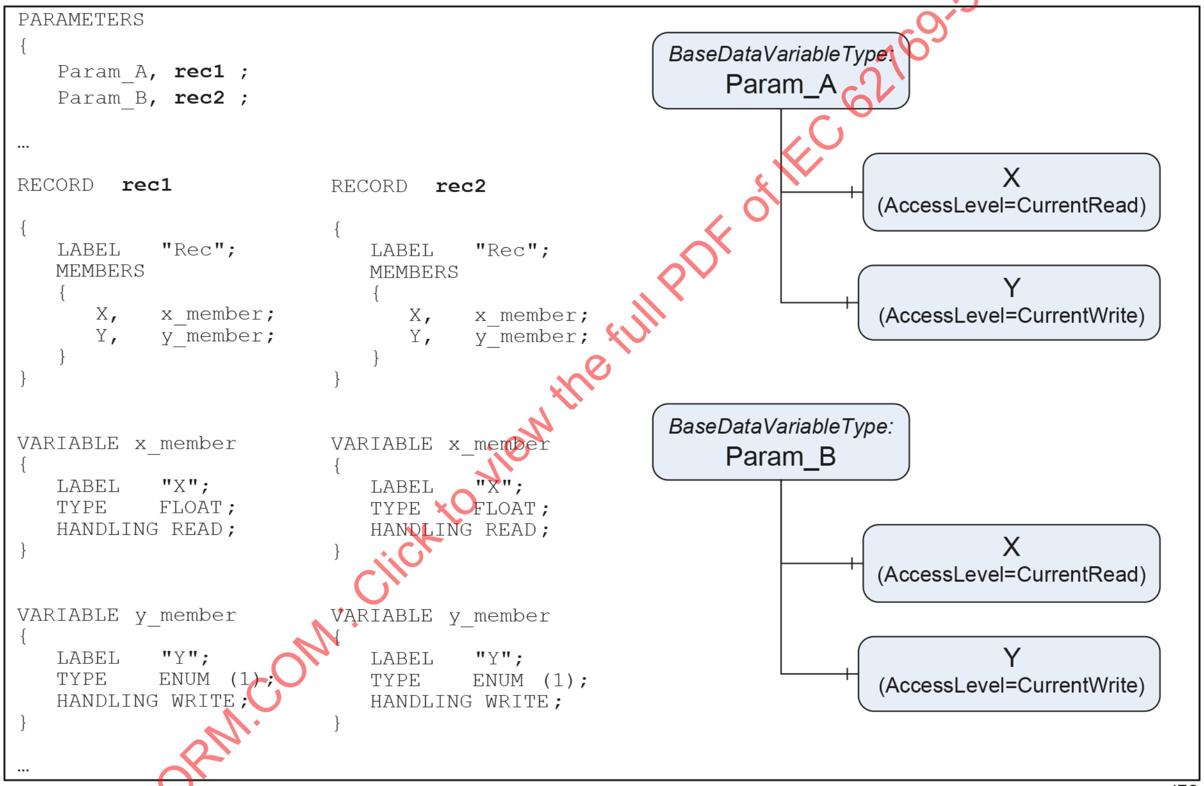
The Boolean array of the DataVariable reflects the status of all bits. TRUE is used when a bit is set and FALSE when a bit is reset.

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.

15.6.7 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 21 for an example of how records are represented in the OPC UA AddressSpace.



```
PARAMETERS
{
   Param_A, rec1 ;
   Param_B, rec2 ;

…

RECORD   rec1                RECORD   rec2

{                            {
   LABEL    "Rec";              LABEL    "Rec";
   MEMBERS                      MEMBERS
   {                            {
      X,    x_member;              X,    x_member;
      Y,    y_member;              Y,    y_member;
   }                            }
}                            }


VARIABLE  x_member           VARIABLE  x_member
{                            {
   LABEL    "X";                LABEL    "X";
   TYPE     FLOAT;              TYPE     FLOAT;
   HANDLING READ;               HANDLING READ;
}                            }

VARIABLE  y_member           VARIABLE  y_member
{                            {
   LABEL    "Y";                LABEL    "Y";
   TYPE     ENUM  (1);          TYPE     ENUM  (1);
   HANDLING WRITE;              HANDLING WRITE;
}                            }

…
```

BaseDataVariableType:
Param_A
X
(AccessLevel=CurrentRead)
Y
(AccessLevel=CurrentWrite)
BaseDataVariableType:
Param_B
X
(AccessLevel=CurrentRead)
Y
(AccessLevel=CurrentWrite)
IEC

Figure 21 – 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 21, 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.

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

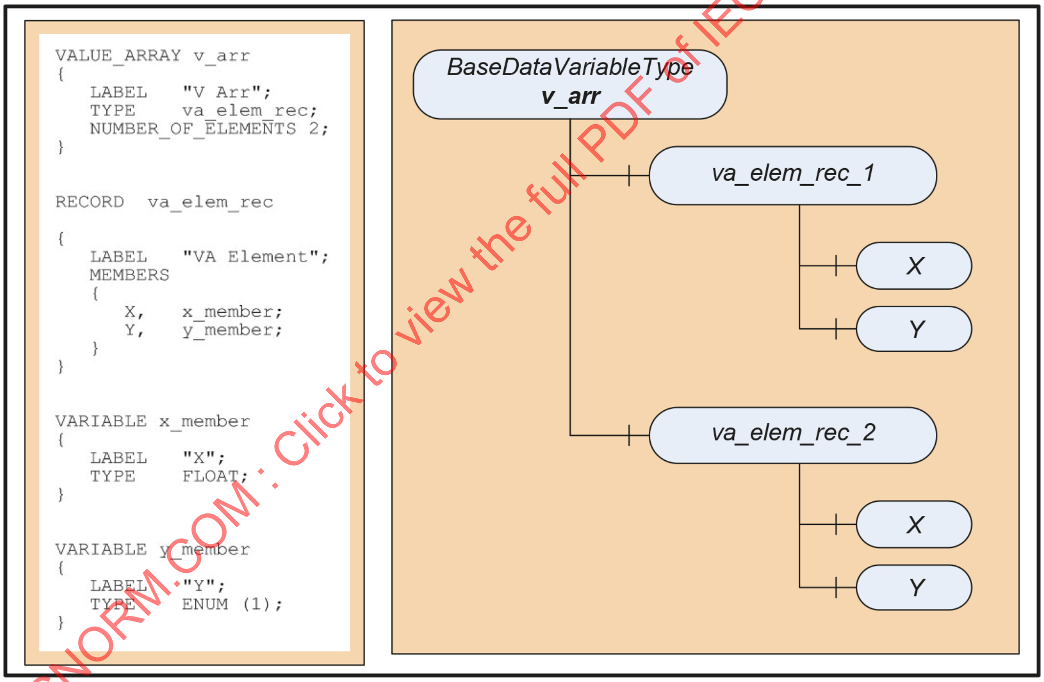
A single DataVariable will represent an EDDL VALUE_ARRAY or LIST item when the data type of the referenced array element has a simple data type.

The OPC UA DataVariable Attributes are set as follows:

- DataType is set to the type of the array element (see Table 50 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 ArrayDimensions Attribute. In the case of an EDDL LIST, the ArrayDimensions Attribute is not exposed. The number of elements is unspecific since the size can change dynamically.

15.6.9 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 22 shows the EDDL sample code of a VALUE_ARRAY of RECORDs and the corresponding complex variable in the OPC UA AddressSpace.



```
VALUE_ARRAY v_arr
{
    LABEL    "V Arr";
    TYPE     va_elem_rec;
    NUMBER_OF_ELEMENTS 2;
}

RECORD  va_elem_rec

{
    LABEL    "VA Element";
    MEMBERS
    {
       X,    x_member;
       Y,    y_member;
    }
}

VARIABLE x_member
{
    LABEL    "X";
    TYPE     FLOAT;
}

VARIABLE y_member
{
    LABEL    "Y";
    TYPE     ENUM (1);
}
```

Figure 22 – 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.

15.6.10 Representation of COLLECTION and REFERENCE ARRAY

The EDDL constructs COLLECTION and REFERENCE ARRAY are mapped to Functional Group Objects (see 15.7) when used for grouping only.

15.6.11 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.

15.6.12 EDDL CLASS Attributes on Parameters

The EddlDictionary Object is a dictionary with the purpose of representing EDDL CLASS Attributes for Parameters. "HasDictionaryEntry" references are used to associate a parameter with EDDL CLASS attributes.

Note that only these EDDL CLASS attributes are mapped to the FDI® Information Model, that are used in a consistent way by EDDL implementations.

The OPC UA types and objects are defined in Table 51, Table 52, and Table 53.

Table 51 – Definition of EddlDictionaryType

Attribute	Value				
BrowseName	**EddlDictionaryEntryType**				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
SubType of DictionaryEntry defined in OPC UA for Devices (DI) Part 100					

Table 52 – Definition of EddlDictionary Object

Attribute	Value				
BrowseName	EddlDictionary				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Component of the DataDictionaries Object defined in IEC 62541-100					
HasTypeDefinition	ObjectType	DictionaryFolderType			
HasComponent	Object	ParameterClass		DictionaryFolderType	

Table 53 – Definition of Parameter Class Attributes

Attribute	Value				
BrowseName	**ParameterClass**				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasTypeDefinition	ObjectType	DictionaryFolderType			
HasComponent	Object	Alarm		EddIDictionaryEntryType	
HasComponent	Object	Contained		EddIDictionaryEntryType	
HasComponent	Object	Correction		EddIDictionaryEntryType	
HasComponent	Object	Device		EddIDictionaryEntryType	
HasComponent	Object	Diagnostic		EddIDictionaryEntryType	
HasComponent	Object	Dynamic		EddIDictionaryEntryType	
HasComponent	Object	Input		EddIDictionaryEntryType	
HasComponent	Object	Local		EddIDictionaryEntryType	
HasComponent	Object	Operate		EddIDictionaryEntryType	
HasComponent	Object	Output		EddIDictionaryEntryType	
HasComponent	Object	Service		EddIDictionaryEntryType	
HasComponent	Object	Specialist		EddIDictionaryEntryType	
HasComponent	Object	Temporary		EddIDictionaryEntryType	
HasComponent	Object	Tune		EddIDictionaryEntryType	

To illustrate the model, an example is shown in Figure 23.

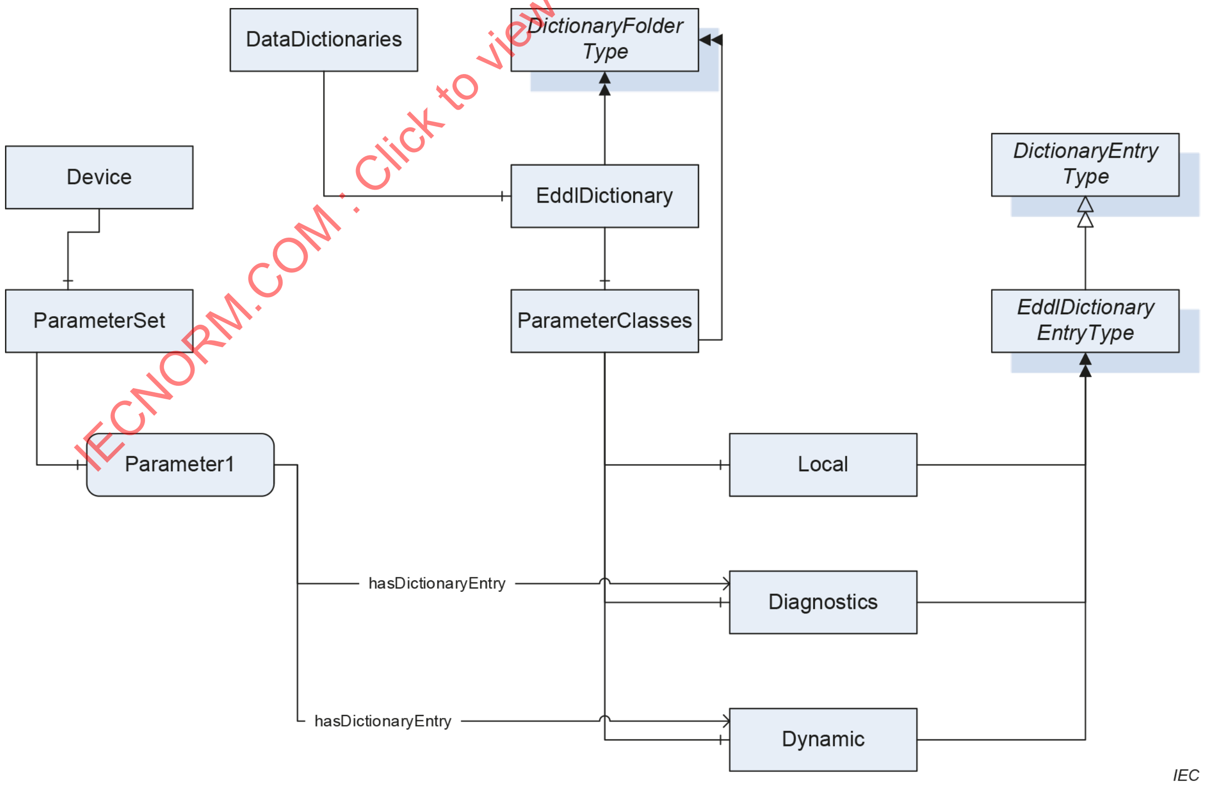


Figure 23 – Example of EDDL CLASS Attributes in the FDI® OPC UA Information Model

15.7 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 building 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.

15.8 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.

15.9 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~~ can be declared private using the PRIVATE Attribute specified in IEC 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.

15.10 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.

15.11 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.

15.12 Semantic Identifies

Semantic identifiers based on the identification schema ISO/IEC 27005 are mapped to EDDL constructs (e.g. COLLECTION, VARIABLE) by the EDDL construct SEMANTIC_MAP (see IEC 61804-3). In the OPC-UA address space, the semantic identifiers are represented by concrete object types derived from the abstract type DictionaryEntryType. The concrete reference type HasDictionaryEntry binds the semantic object to the OPC-UA object representing the EDDL construct (e.g. COLLECTION, VARIABLE). SEMANTIC_MAP can also be used to map enumeration items to semantic identifiers (e.g. units).

The EDDL SEMANTIC_MAP construct maps a list (one or more) keys to a list (one or more) values. The keys are data dictionary ids and the values are EDDL item identifier.

A DictionaryEntryType Object is created for each key term (data dictionary id) of the EDDL SEMANTIC_MAP. The key value is mapped to the BrowseName and NodeId property of the DictionaryEntryType Object.

A HasDictionaryEntry reference is created for each value listed in the EDDL SEMANTIC_MAP construct with the SourceNode set to the NodeId of the OPC UA object representing EDDL item identified.

15.13 DictionaryIds Property

For enumerated Variables or Enumeration DataTypes (see IEC 61804-3), the enumeration values may have a specific reference to a dictionary entry. To express this, Enumeration and OptionSet DataTypes or Variables with VariableTypes like MultiStateDiscreteType, MultiStateValueDiscreteType or OptionSetType reference an additional Property with the BrowseName DictionaryIds which provides a Value with an array of DataType NodeId where each array item contains the NodeId of the corresponding DictionaryEntryType Object. The DictionaryIds Property is formally defined in Table 54.

Table 54 – DictionaryIds Definition

Attribute	Value
BrowseName	DictionaryIds
IsAbstract	False
TypeDefinition	PropertyType
DataType	NodeId[]

15.14 MultiStateDictionaryEntryDiscreteType

This VariableType defines the general characteristics of a DiscreteItem that can have more than two states including semantic definitions. The MultiStateDictionaryEntryDiscreteType derives from the MultiStateDiscreteType. It is formally defined in Table 55.

Table 55 – MultiStateDictionaryEntryDiscreteType definition

Attribute	Value				
BrowseName	MultiStateDictionaryEntryDiscreteType				
NodeClass	VariableType				
ValueRank	Scalar				
DataType	UInteger				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the MultiStateDiscreteType defined in IEC 62541-8; i.e the Properties of that type are inherited.					
HasProperty	Variable	EnumDictionary Entries	NodeId[][]	PropertyType	Optional
HasProperty	Variable	ValueAsDictionary Entries	NodeId[]	PropertyType	Mandatory

EnumDictionaryEntries is a two-dimensional array of NodeIds. The first dimension is used to list all possible dictionary entry values for the related variable in a specific dictionary (e.g. CDD or eCl@ss). The second dimension is used to reference this dictionary. The size of the first array dimension shall be the same as the size of EnumStrings.

ValueAsDictionaryEntries provides a list of all dictionary entry values in the different dictionaries related to the current value of the variable.

If an instance of this type is writeable, the property ValueAsDictionaryEntries shall be writeable as well. Clients are allowed to write an array with only one entry and the server is responsible to resolve additional appropriate entries. This will have the same result as writing the value attribute, but the client does not need knowledge about the numeric values.

The NodeIds represent the dictionary entries and can be generated with dictionary knowledge.

15.15 GetNodeIdsByDictionaryEntryId

GetNodeIdsBySemanticId is a custom query used to get the NodeIds for nodes that have a HasDictionaryEntry Reference to a DictionaryEntryType Object matching the provided DictionaryEntryId. The Method returns only nodes that are referenced by the StartingNode with the defined ReferenceType.

Signature

```
GetNodeIdsByDictionaryEntryId (
    [in] NodeId StartingNode
    [in] NodeId ReferenceType
    [in] NodeId DictionaryEntryId
    [out] NodeId[] Nodes
  );
```

The Method arguments and result codes are described in Table 56 and Table 57.

Table 56 – GetNodeIdsByDictionaryEntryId Method arguments

Argument	Description
StartingNode	A NodeId defining the starting node for the query.
ReferenceType	A NodeId defining the ReferenceType to take into account for the query.
DictionaryEntryId	A NodeId defining the DictionaryEntry Object that represents the queried dictionary entry.
Nodes	Contains an Array of NodeIds of the nodes that match the provided query parameters.

Table 57 – GetNodeIdsByDictionaryEntryId Method result codes

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.
Bad_NotWritable	See IEC 62541-4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See IEC 62541-4 for a general description. File was not opened for write access.

Table 58 specifies the AddressSpace representation for the GetNodeIdsByDictionaryEntryId Method.

Table 58 – GetNodeIdsByDictionaryEntryId

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

16 Profiles

This chapter defines the profiles and conformance units for the OPC UA Information Model for FDI® Clients and FDI® Servers. Profiles are named groupings of conformance units. Facets are profiles that will be combined with other Profiles to define the complete functionality of an OPC UA Server or Client.

FDI® Servers (FDI® Hosts) that implement a subset of the FDI® Information Model that is relevant for generic OPC UA Clients shall implement as a minimum the BaseDevice_Server_Facet specified in IEC 62541-100 (OPC UA for Devices).

Table 59 specifies the facet for FDI® Servers (FDI® Hosts) that support FDI® Clients.

Table 59 – FDI® Server Facet Definition

Conformance Unit	Description	Optional/ Mandatory
FDI® Information Model	Support Objects that conform to the Types specified in this document and in IEC 62541-100.	M

Table 60 defines the facet for FDI® Clients that use the FDI® Information Model.

Table 60 – FDI® Client Facet Definition

Conformance Unit	Description	Optional/ Mandatory
FDI® Client Information Model	Consume Objects that conform to the Types specified in this document and in IEC 62541-100.	M

Annex A
(normative)

Namespace and Mappings

This annex defines the numeric identifiers for all of the numeric NodeIds defined in this document. 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.

An electronic version of the complete Information Model defined in this document 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 is in a separate file with name:

OPCUA_Part5_Model_V1.1\Opc.Ua.FDI5.NodeSet2.xml

Bibliography

IEC 61987 (all parts), *Industrial-process measurement and control – Data structures and elements in process equipment catalogues*

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

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

~~IEC 62769-1xx (all parts), *Field Device Integration (FDI) – Profiles*~~

ISO/IEC 27005, *Information security, cybersecurity and privacy protection − Guidance on managing information security risks*

NAMUR NE107, *Self-Monitoring and Diagnosis of Field Devices*
<available at www.namur.de > [viewed 2023-02-02]

IEC 62769-5

Edition 3.0 2023-04

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

**Field device integration (FDI®) –
Part 5: FDI Information Model**

**Intégration des appareils de terrain (FDI®) –
Partie 5: Modèle d'Information FDI**

IEC 62769-5:2023-04(en-fr)

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

FIELD DEVICE INTEGRATION (FDI®) –

Part 5: FDI® Information Model

FOREWORD

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. It is an International Standard.

This third edition cancels and replaces the second edition published in 2021. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) added INTERACTIVE_TRANSFER_TO_DEVICE ACTION.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65E/858/CDV	65E/915/RVC

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

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 document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

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

IMPORTANT – The "colour inside" logo on the cover page of this document 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.

FIELD DEVICE INTEGRATION (FDI®) –

Part 5: FDI® Information Model

1 Scope

This part of IEC 62769 defines the FDI®[1] 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.

[1] FDI® is a registered trademark of the non-profit organization Fieldbus Foundation, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

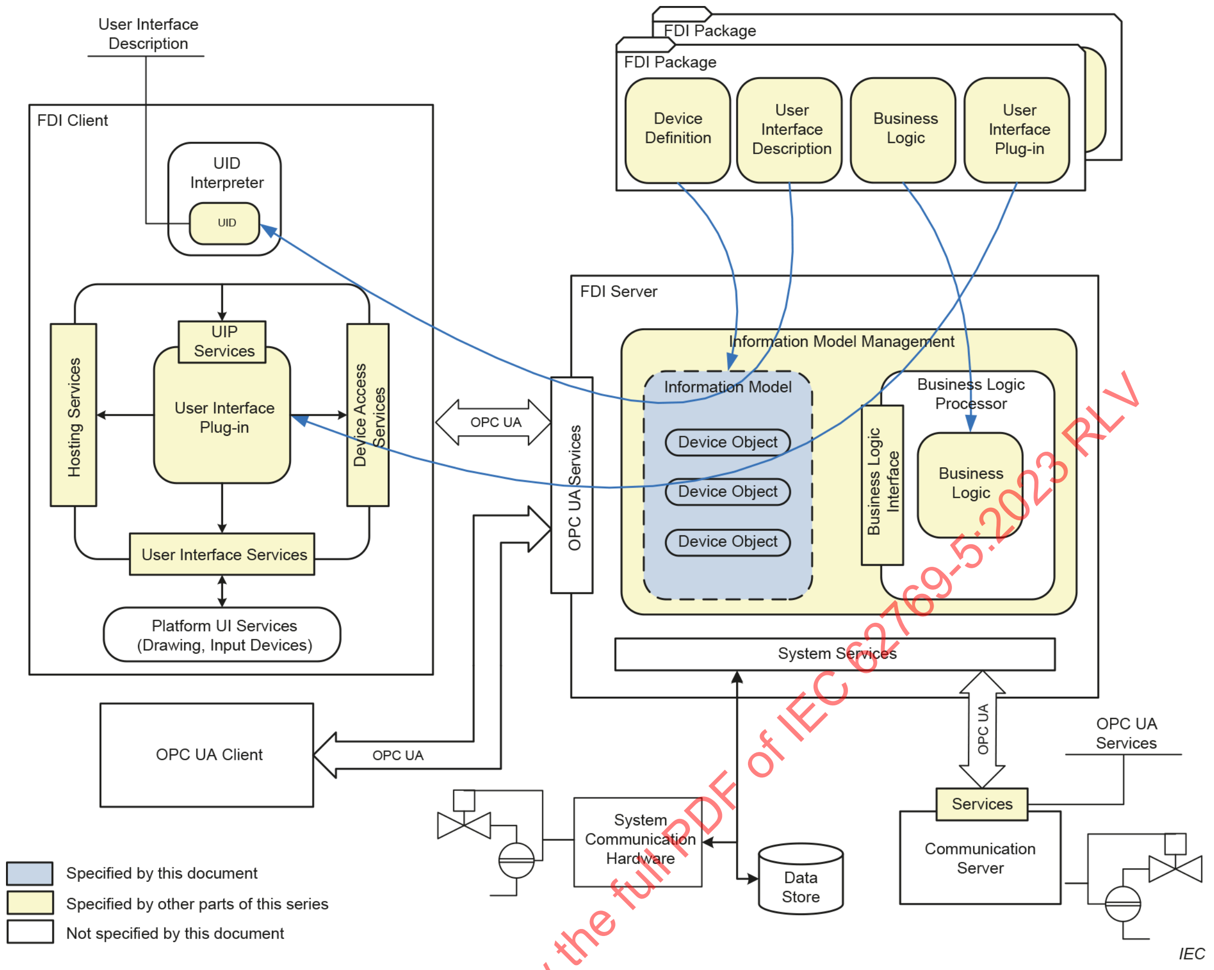


Figure 1 – FDI® architecture diagram

2 Normative references

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

IEC 61784-1-3:2023, *Industrial networks – Profiles – Part 1-3: Fieldbus profiles – Communication Profile Family 3*

IEC 61804-3, *Devices and integration in enterprise systems − Function blocks (FB) for process control and electronic device description language (EDDL) − Part 3: EDDL syntax and semantics*

IEC 61804-4, *Devices and integration in enterprise systems − Function blocks (FB) for process control and electronic device description language (EDDL) − Part 4: EDD interpretation*

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, *OPC Unified Architecture – Part 100: Device Interface*

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

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

IEC 62769-3, *Field Device Integration (FDI®) – Part 3: Server*

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

IEC 62769-6, *Field Device Integration (FDI®) – Part 6: FDI® Technology Mappings*

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

IEC 62769-1xx (all parts), *Field Device Integration (FDI®) – Part 1xx-y: Profiles*

OPC 10000-19, *OPC Unified Architecture – Part 19: Dictionary Reference*

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 and IEC 62769-3 apply.

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

- IEC Electropedia: available at https://www.electropedia.org/
- ISO Online browsing platform: available at https://www.iso.org/obp

3.2 Abbreviated terms and acronyms

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

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

3.3 Conventions

3.3.1 Capitalization

Capitalization of the first letter of words is used in the IEC 62769 series to emphasize an FDI® defined term.

3.3.2 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 document. NodeClasses representing types always have a shadow.

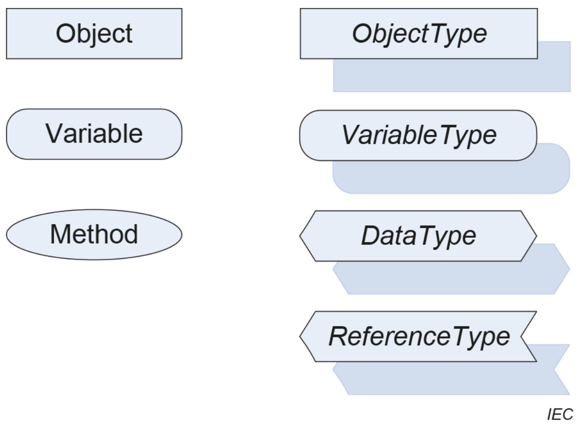


Figure 2 – OPC UA graphical notation for NodeClasses

Figure 3 shows the symbols for the ReferenceTypes used in this document. 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.

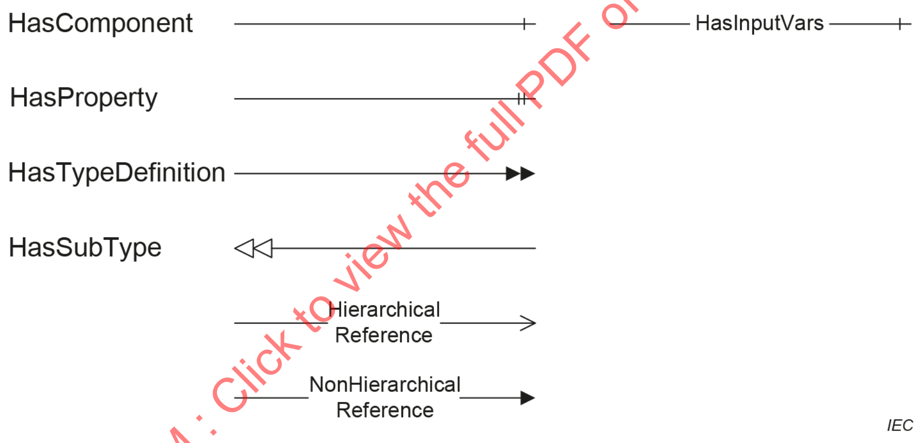


Figure 3 – OPC UA graphical notation for References

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).

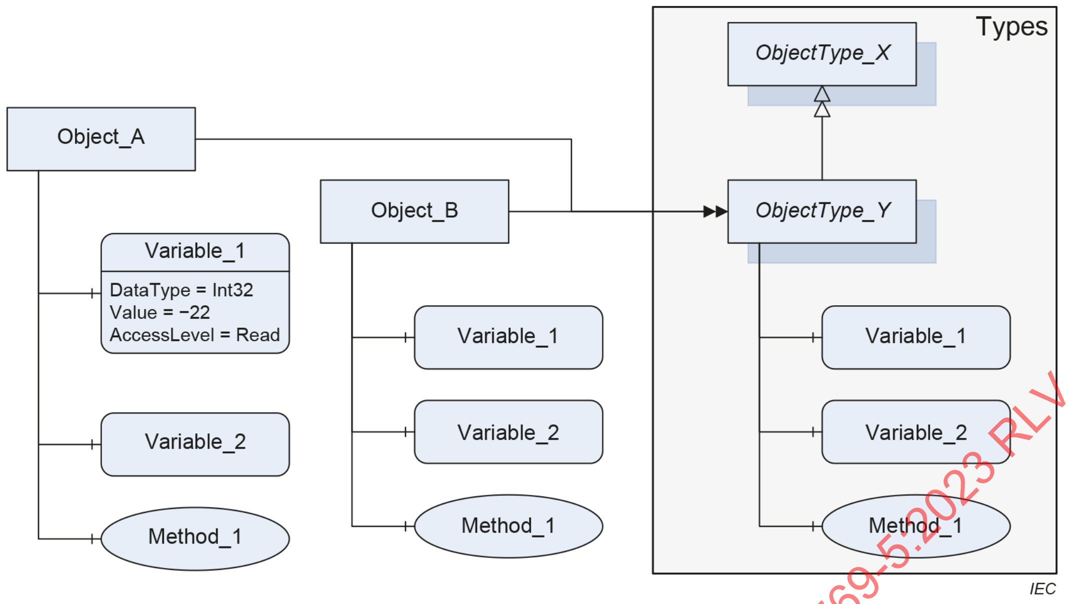


Figure 4 – OPC UA graphical notation example

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.

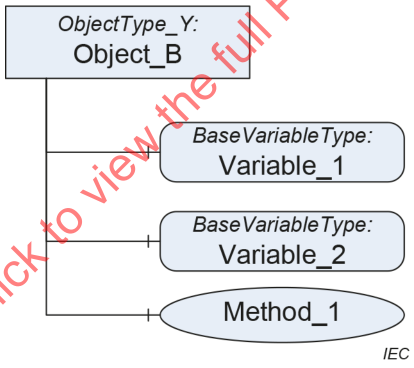


Figure 5 – Optimized Type Reference

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 can have component DataVariables. This is typically used by Servers to expose individual elements of arrays and structures. This document 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. It is a DeviceType Object that is a subtype of TopologyElementType and inherits all components of this type. The component ParameterSet contains all Variables describing the Device. The component MethodSet contains all Methods provided by the Device. Components of the FunctionalGroupType are used to collect 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.

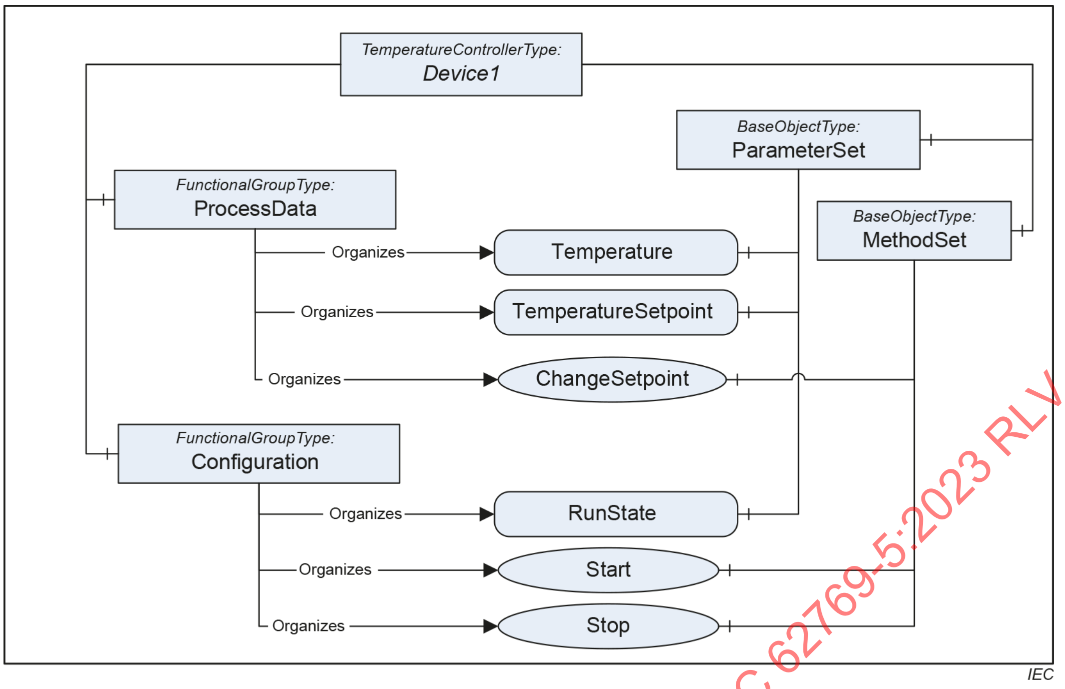


Figure 6 – OPC UA Devices example: Functional Groups

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.

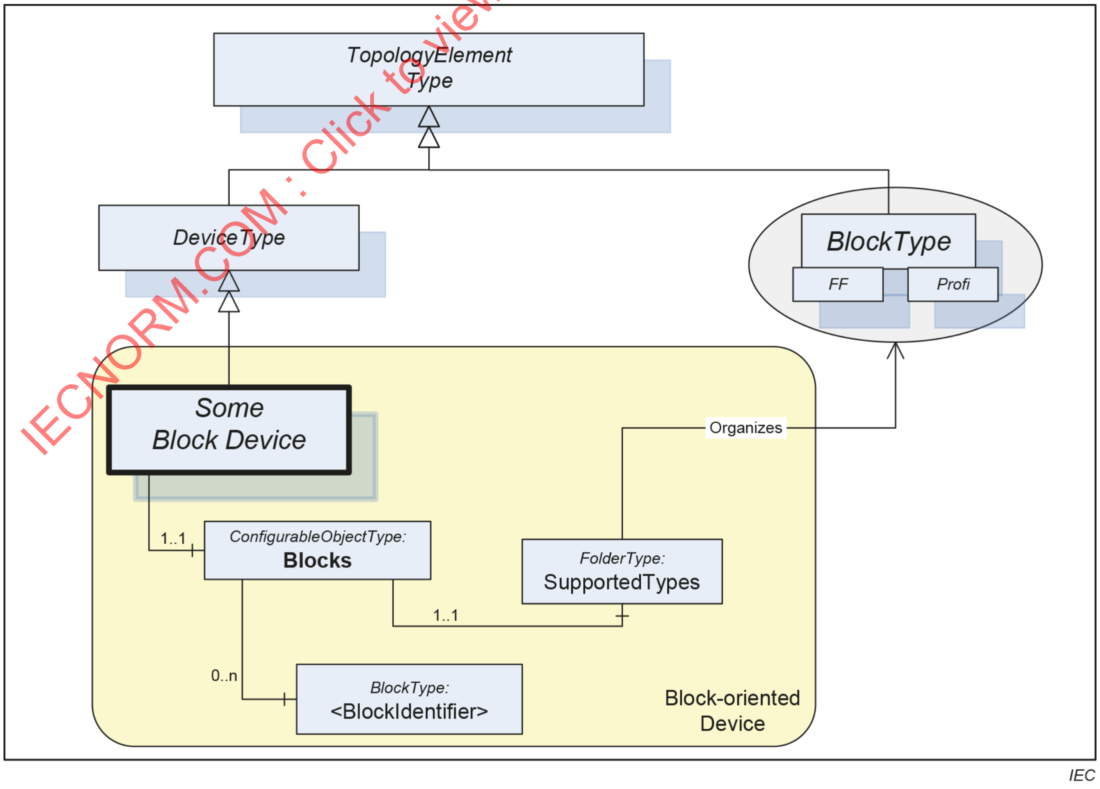


Figure 7 – OPC UA Devices example: Configurable components

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 can 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).

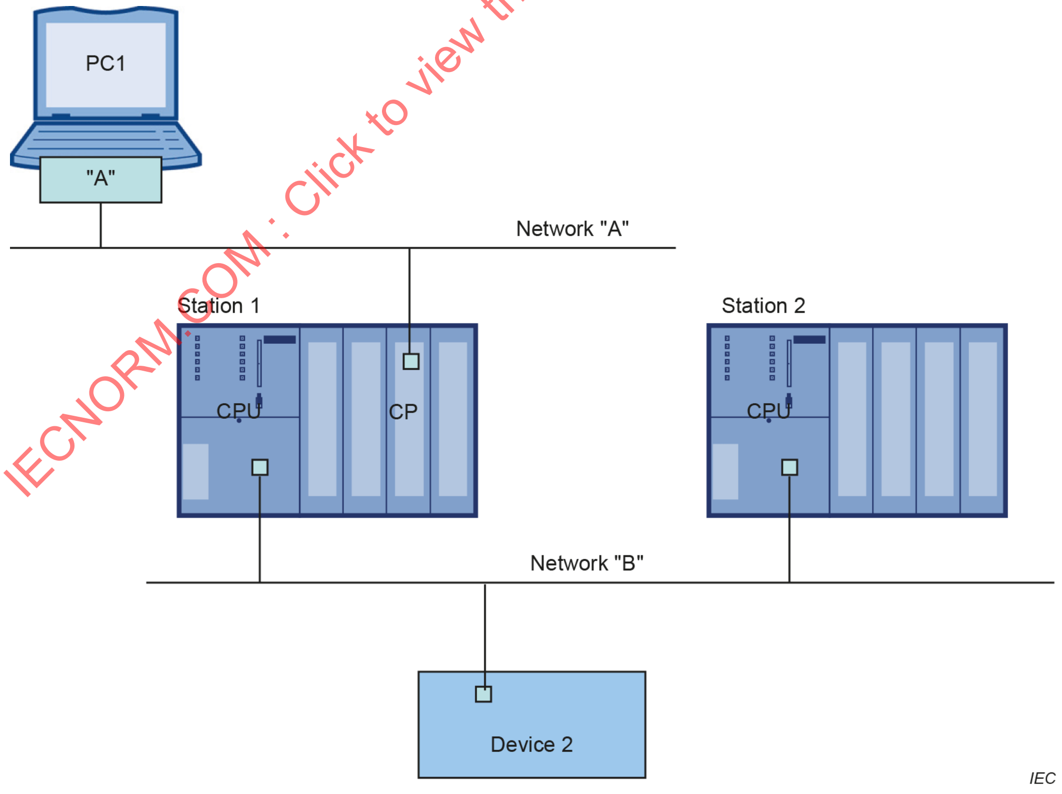


Figure 8 – Example of an automation system

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.

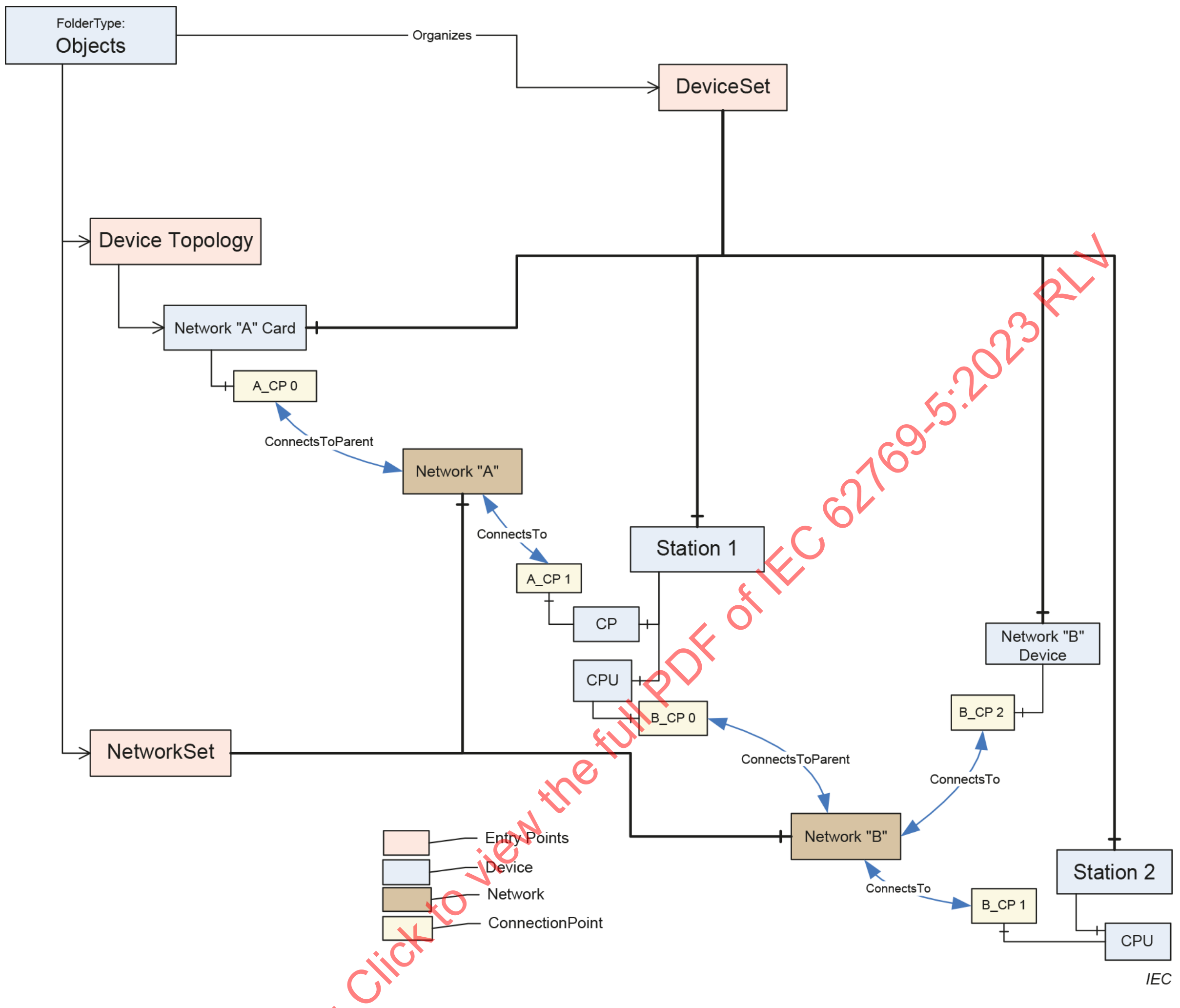


Figure 9 – Example of a Device topology

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 can exist in the AddressSpace.

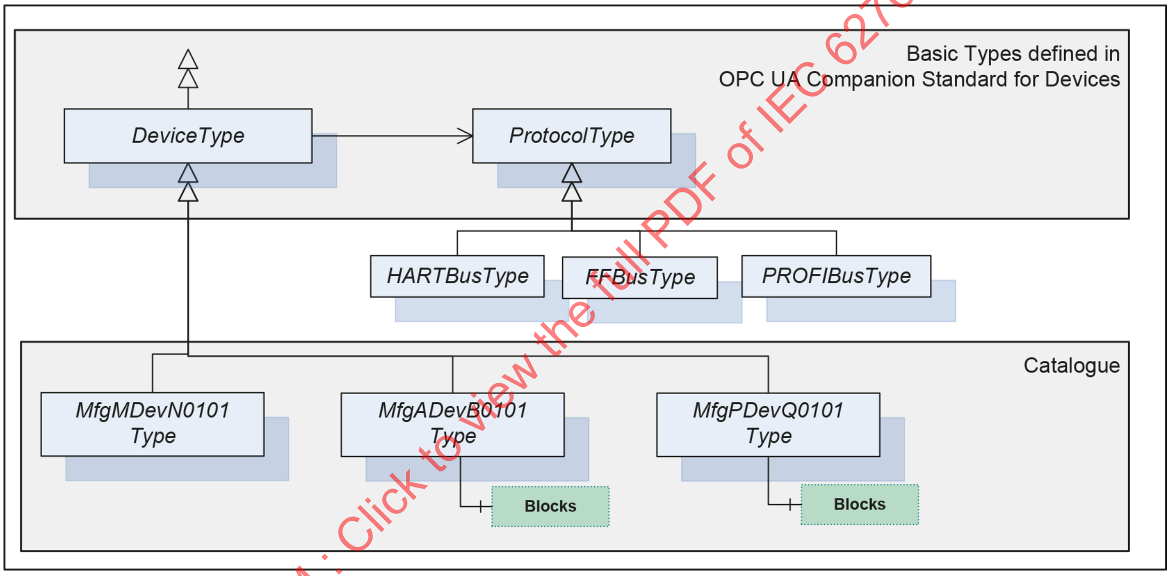


Figure 10 – Example Device Types representing a catalogue

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 shall go through Station 1, which provides the communication services for the PROFIBUS DP Network (see IEC 61784-1-3, CPF 3). This can be achieved through Nested Communication, which is specified in IEC 62769-3. Communication Devices and Communication Servers are specified in IEC 62769-7.

5.6 Semantic Information

The Type System of OPC UA already provides means to express the semantic of an Object. As an example, The Devices Companion Specification defines the DeviceType ObjectType expressing that instances of this ObjectType represent devices. However, the detailed semantic of the device is not specified further. Semantic information provides a means to represent that an Object has the semantic differential pressure transmitter for instance. This allows clients to automatically retrieve and identify such devices.

Objects in OPC UA typically have Variables holding variable values. While the BrowseName or DisplayName provide a "hint" regarding the semantic of the Variable, this cannot be used to automatically identify the semantic of a Variable. As an example, the ParameterSet of a DeviceType Object contains a flat list of the parameters of the device. Semantic information provides means to represent that a Variable (e.g. in the ParameterSet) has a specific semantic associated with it. This allows clients to automatically retrieve and identify such Variables.

The DictionaryEntryType Object, defined in OPC-10000-19, is used to add semantic information to objects in the FDI® Information Model. Subtypes of DictionaryEntryType shall define their own namespace (e.g. http://iec.ch/cdd/iec61987). The NodeId of Objects shall consist of the corresponding namespace index and the value of the ID Property as the identifier part. This allows direct access to the Semantic Object without further indirection. Annex A defines the numeric identifiers for all of the numeric NodeIds used in this document.

The OPC UA Server shall create a concrete object type derived from the abstract DictionaryEntryType for each dictionary referenced. Objects such as DeviceType, FunctionalGroupType, or VariableType can reference one or more DictionaryEntryType objects using the HasDictionaryEntry reference to indicate the semantic meaning for the object. The DictionaryId property of the DictionaryEntryType object contains the identifier value of an external dictionary entry. A client can use these references to access device data with only needing to know semantically what information is desired (i.e. specific object BrowseNames are not required).

The following example provides an overview of this design principle (Figure 11). This example uses the IEC Common Data Dictionary (CDD); however, it should be noted that any qualifying dictionary can be used. The IEC Common Data Dictionary (CDD) describes classes and properties with the purpose to characterize equipment. The IEC 61987 series for instance describes process automation equipment. Every class and property provide a set of attributes such as Version, Revision, Preferred name, Definition, etc. Most importantly, it describes a unique identifier for every class of property. In the case of IEC CDD this is an International Registration Data Identifier (IRDI). In this example, IEC61987DictionalEntryType is defined as a subtype of DictionaryEntryType. The Object DifferentialPressureTransmitter of this ObjectType contains the IEC CDD attribute values as its property values. The IRDI is set as the value of the ID property.

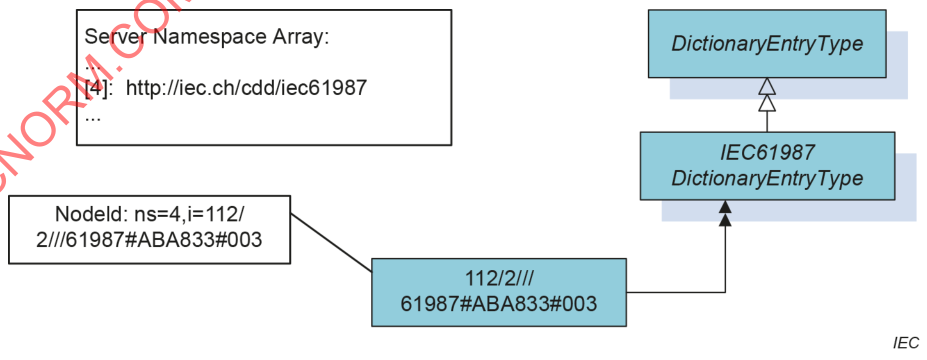

Server Namespace Array:
...
[4]: http://iec.ch/cdd/iec61987
...
DictionaryEntryType
IEC61987 DictionaryEntryType
NodeId: ns=4,i=112/ 2///61987#ABA833#003
112/2/// 61987#ABA833#003
IEC

Figure 11 – Example of concrete DictionaryEntryType and Object

Variables can be enumerations. In such cases, the Variable can have a specific semantic as well as each enumeration item of the Variable. As an example, a Variable can contain the units of the measurement value that can be configured. So, the semantic of the Variable itself is "measurement unit". The enumeration items of that Variable then stand for a specific unit of measure. In case of temperature, the first enumeration item can represent degrees Celsius, the second item represents degrees Fahrenheit, the third item represents degrees Kelvin and so on. Associating enumeration items with a specific semantic allows clients to automatically retrieve information such as the currently configured measurement unit without requiring interment knowledge of the particular unit code being used by the device.

Enumerated Variables contain an additional property DictionaryEntries, defined in 15.13, which is an array of DataType NodeId where each array item contains the NodeId of the corresponding DictionaryEntry Object. Figure 12 shows an example: The Variable Parameter2 has the Property DictionaryEntries. The array items of the DictionaryEntries Property contain the NodeIds of the corresponding DictionaryEntry Objects. In the example, Parameter2[0] is related to the Semantic1 Object, Parameter2[1] is not related to any Semantic Object and Parameter2[2] is related to the Semantic2 Object.

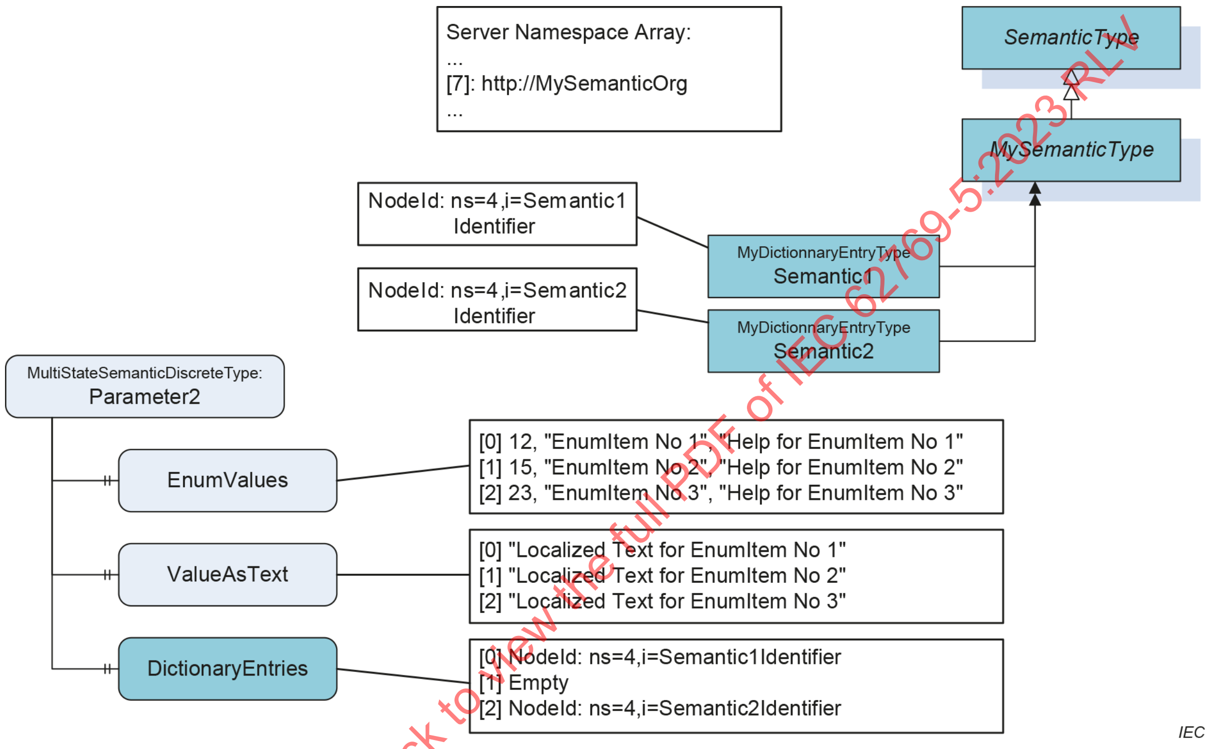

Server Namespace Array:
...
[7]: http://MySemanticOrg
...
SemanticType
MySemanticType
NodeId: ns=4,i=Semantic1 Identifier
NodeId: ns=4,i=Semantic2 Identifier
MyDictionnaryEntryType
Semantic1
MyDictionnaryEntryType
Semantic2
MultiStateSemanticDiscreteType:
Parameter2
EnumValues
ValueAsText
DictionaryEntries
[0] 12, "EnumItem No 1", "Help for EnumItem No 1"
[1] 15, "EnumItem No 2", "Help for EnumItem No 2"
[2] 23, "EnumItem No 3", "Help for EnumItem No 3"
[0] "Localized Text for EnumItem No 1"
[1] "Localized Text for EnumItem No 2"
[2] "Localized Text for EnumItem No 3"
[0] NodeId: ns=4,i=Semantic1Identifier
[1] Empty
[2] NodeId: ns=4,i=Semantic2Identifier
IEC

Figure 12 – Example of DictionaryEntries

The custom query method GetNodeIdByDictionaryEntryId, defined in 15.14, can be used by a client to search for the nodes that are referencing a specified dictionary entry id. This provides an efficient means for finding data using semantic information.

6 AddressSpace organization

To promote interoperability of FDI® Clients and FDI® Servers, a set of Objects and relationships are defined in the following Clause 7. 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:

- The Types-part contains information about all components that have been generated based on descriptive information from FDI® Packages (see 5.4).
- 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 can 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 document) 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 13.

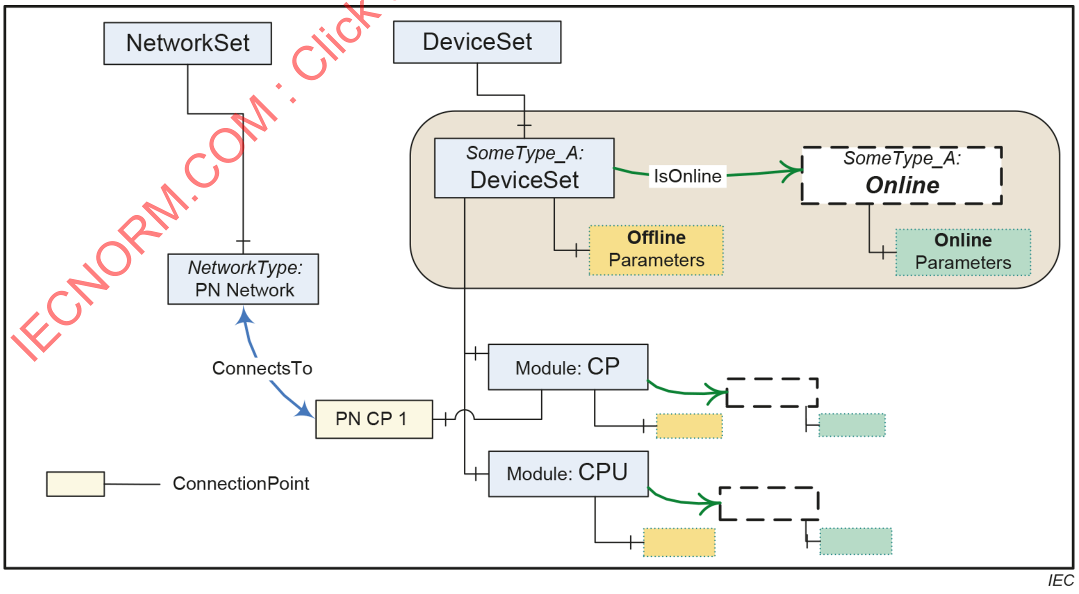


Figure 13 – Online component for access to device data

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

7.3.1 DeviceHealth Mapping

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 can vary from several seconds up to minutes.

The mapping of the GetHealthStatus return values to OPC UA is specified in Table 1.

Table 1 – DeviceHealth Mapping

GetHealthStatus	OPC UA
0 – Indeterminate	Indeterminate is not defined in the the DeviceHealth data type in IEC 62541-100. Servers that cannot determine the health status of the device shall return the OPC UA Read operation for this Property with an appropriate OPC UA status code, for example: Bad_NotConnected Bad_OutOfService Bad_NoCommunication Bad_NotSupported
The following values can be mapped to corresponding values defined for the DeviceHealth data type in IEC 62541-100.	
GetHealthStatus	**DeviceHealth data type values**
1 – Failure	FAILURE_1
2 – Function Check	CHECK_FUNCTION_2
3 – Out of Specifications	OFF_SPEC_3
4 – Maintenance Required	MAINTENANCE_REQUIRED_4
5 – Good	GOOD_0

Different to OPC UA for Devices (IEC 62541-100), support of the DeviceHealth Property is mandatory for Device Objects in FDI®. This is illustrated in Table 2. The complete DeviceType model and its formal definition are in IEC 62541-100.

Table 2 – DeviceType definition (excerpt applicable for Subclause 7.3.1)

Attribute	Value				
BrowseName	DeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the TopologyElementType defined in IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	~~Optional~~ → Mandatory
………					

7.3.2 DeviceHealth Diagnostics

In certain cases, a Device can have additional information associated with the health status, e.g. the possible cause of an abnormal DeviceHealth status and suggested actions to return to normal.

This additional information is available with the DeviceHealthDiagnostics Variable. It is formally defined in Table 3.

Table 3 – DeviceType definition with DeviceHealth and DeviceHealthDiagnostics

Attribute	**Value**				
BrowseName	DeviceType				
IsAbstract	True				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the TopologyElementType defined in IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	Mandatory
HasComponent	Variable	DeviceHealthDiagnostics[]	LocalizedText	BaseDataVariableType	Mandatory
………					

DeviceHealthDiagnostics is an array of LocalizedText. Each array element contains related diagnostic information. The language shall match the requested locale specified in the OPC UA Session. See 15.2.1 on how to select the proper language for the current OPC UA Session.

When DeviceHealth is requested, the Server shall always also fetch the matching DeviceHealthDiagnostics. The Server shall assure that the values are synchronized by executing GetHealthStatus first and then reading the DeviceHealthDiagnostics. The Server will cache the DeviceHealthDiagnostics as part of the OPC UA Session until DeviceHealth is requested again.

If DeviceHealthDiagnostics is read with a separate service request, the Server shall return the diagnostic information associated with the most recently read DeviceHealth status. If the DeviceHealth status has not been read in the current OPC UA Session, the Server shall return Bad_NotReadable.

If no diagnostic information is available, the returned Value is Null.

See IEC 62769-4 on how DeviceHealthDiagnostics maps to the corresponding EDD information.

Example of a DeviceHealthInformation value with two array elements:

- [0]:
  "Critical Power Failure\n
  Possible cause: Critical Power failure has occurred\n
  Suggested action: Please check device/surroundings/process.\n"
- [1]:
  "Device Malfunction\n
  Possible cause: The device has detected a serious hardware error or failure.\n
  Suggested action: Check detailed device diagnosis if possible and/or replace device hardware if necessary.\n"

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 14 illustrates the type hierarchy of the user interface elements defined in IEC 62541-100 and in this document.

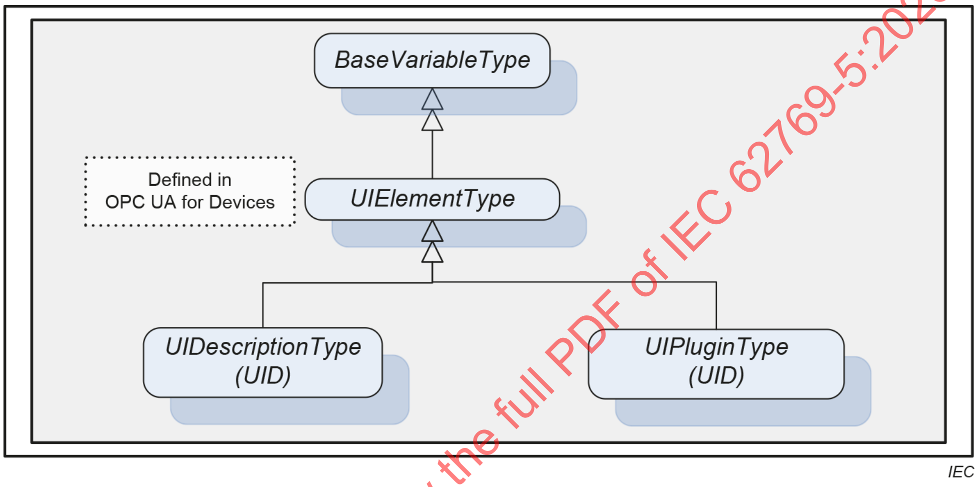


Figure 14 – Hierarchy of user interface Types

7.4.2 UI Description Type

FDI® Servers can 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 4.

Table 4 – 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 5.

Table 5 – 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 can 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 might 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 can be large. Reading large values with a single Read operation might 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 can 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 can 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 15 illustrates how Actions are integrated into a TopologyElement.

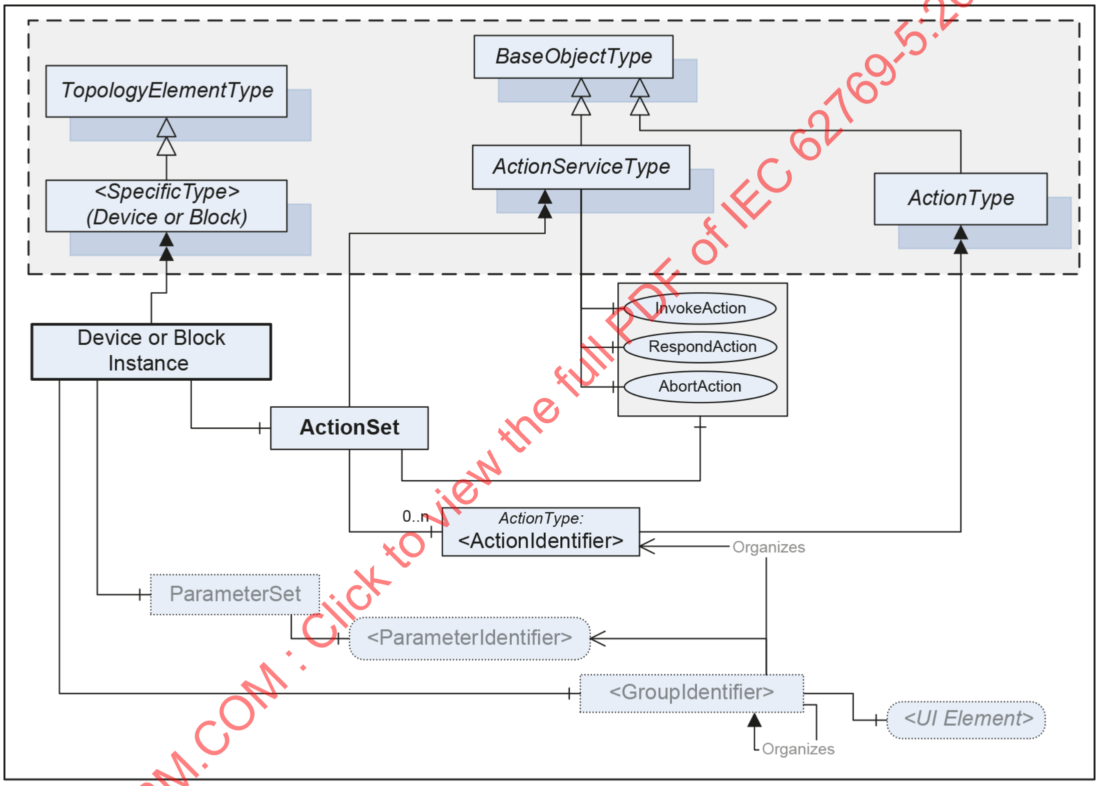


Figure 15 – Integration of Actions within a TopologyElement

ActionSet is used to aggregate the Actions for a specific TopologyElement. This Object is not available on the Type and is only available in an instance if Actions exist for this TopologyElement.

Actions can also be referenced from FunctionalGroup Objects.

7.6.2 Action Type

This ObjectType defines the structure of an Action. It is formally defined in Table 6.

Table 6 – 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 7. Its use is illustrated in Figure 15.

Table 7 – 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 document. 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 16.

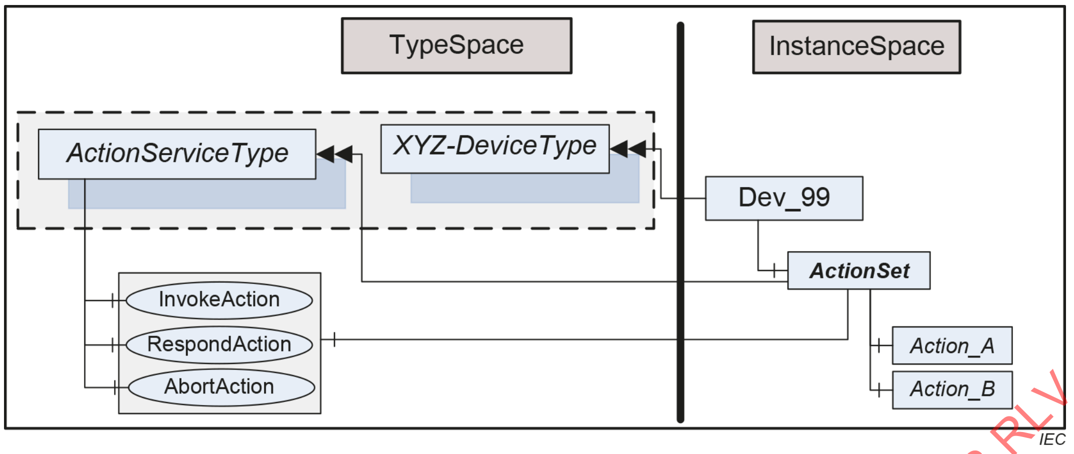


Figure 16 – Action Service

This Object is used as container for the ActionService Methods and shall have the BrowseName ActionSet. It is formally defined in Table 7. 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 has a chance to subscribe.

The signature of this Method is specified below. Table 8 and Table 9 specify the arguments and AddressSpace representation, respectively.

Signature

```
InvokeAction(
  [in]  String      ActionName,
  [in]  String      MethodArguments,
  [out] NodeId      ActionNodeId,
  [out] Int32       InvokeActionError);
```

Table 8 – 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 9 – 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 10 and Table 11 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 10 – 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 11 – 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 12 and Table 13 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 12 – 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 13 – 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

7.6.8 Interactive Transfer to device

The functionality to transfer data to the online Device including user interaction is provided as INTERACTIVE_TRANSFER_TO_DEVICE ACTION by the FDI® Server. This action has no arguments. It is started using the InvokeAction method.

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 62769-3. 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 can 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 14.

Table 14 – 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 17.

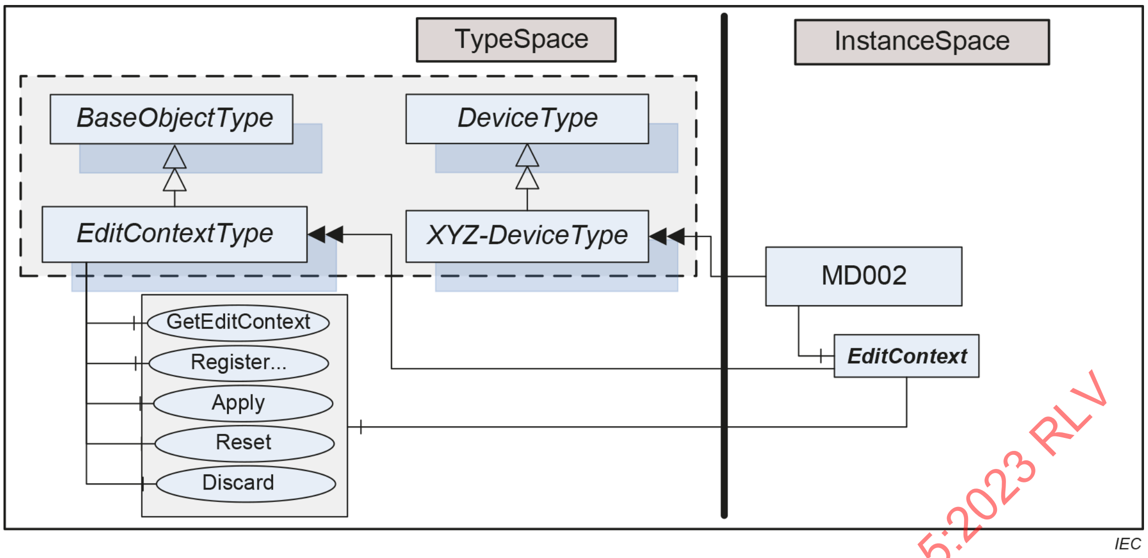


Figure 17 – EditContext type and instance

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 can 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 15 and Table 16 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 15 – 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 16 – 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 17 and Table 18 specify the arguments and AddressSpace representation, respectively.

Signatures

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

Table 17 – 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 18 – 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 19 – 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 20.

Table 20 – 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 offline representation of the Node in the EditContext. See IEC 62541-3 for details.
offlineDeviceNodeId	NodeId	This NodeId shall be used to address the offline 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 21 and Table 22 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 21 – 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 22 – 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 23.

Table 23 – 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 24 and Table 25 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 24 – 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 25 – 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 26 and Table 27 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 26 – 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 27 – 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 DirectDeviceAccess

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 can 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 18 shows the DirectDeviceAccessType definition. It is formally defined in Table 28.

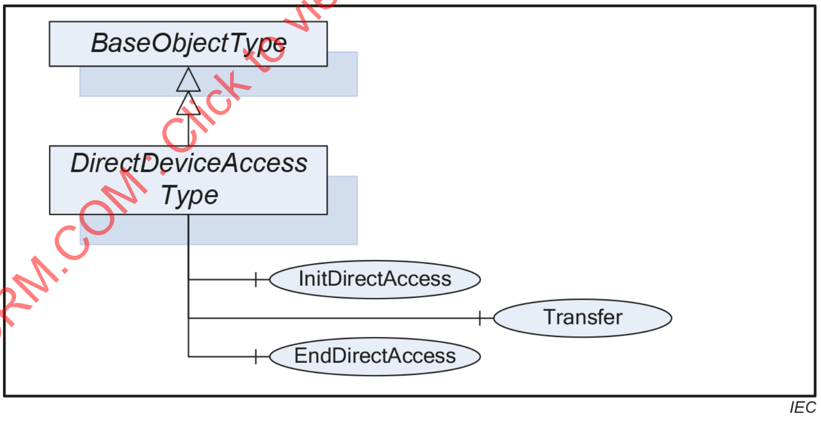


Figure 18 – DirectDeviceAccessType

Table 28 – DirectDeviceAccessType Definition

Attribute	Value				
BrowseName	Discard				
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 document. 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 19.

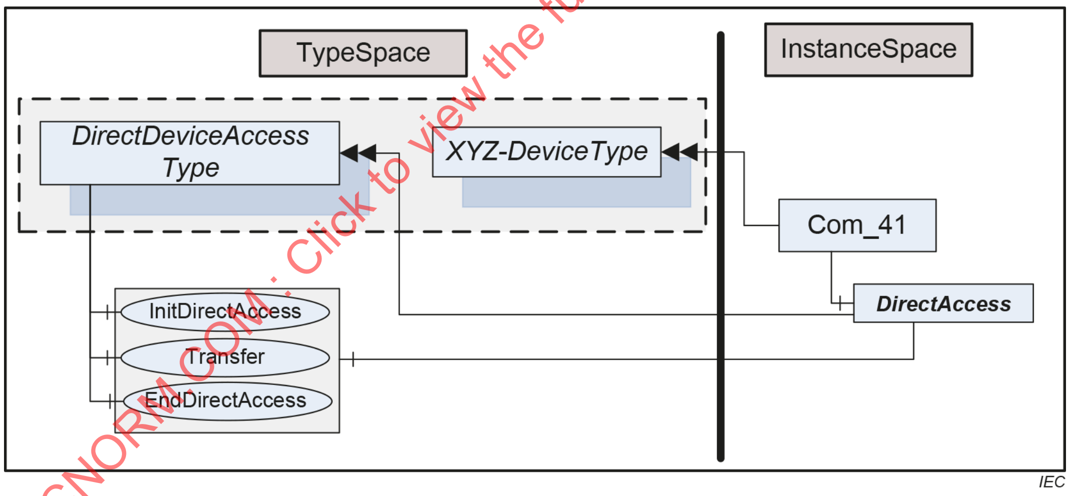


Figure 19 – DirectDeviceAccess instance

This Object is used as container for the DirectDeviceAccess Methods and shall have the BrowseName DirectAccess. It is formally defined in Table 29. 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 29 – 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 30 and Table 31 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 30 – 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 cannot be directly accessed -2 – E_LockRequired – the element is not locked as required -3 – E_InvalidState – the device is already in DirectAccess mode

Table 31 – 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 32 and Table 33 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 32 – 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 33 – 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 34 and Table 35 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 34 – 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 and IEC 62769-1xx series.
ReceiveData	XML document based on the TransferResultDataType as specified in the communication profile-specific XML schema. See IEC 62769-4 and IEC 62769-1xx series.
TransferError	0 – OK -1 – E_InvalidState – the device is not in DirectAccess mode.

Table 35 – 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 behaviours 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 20. 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.

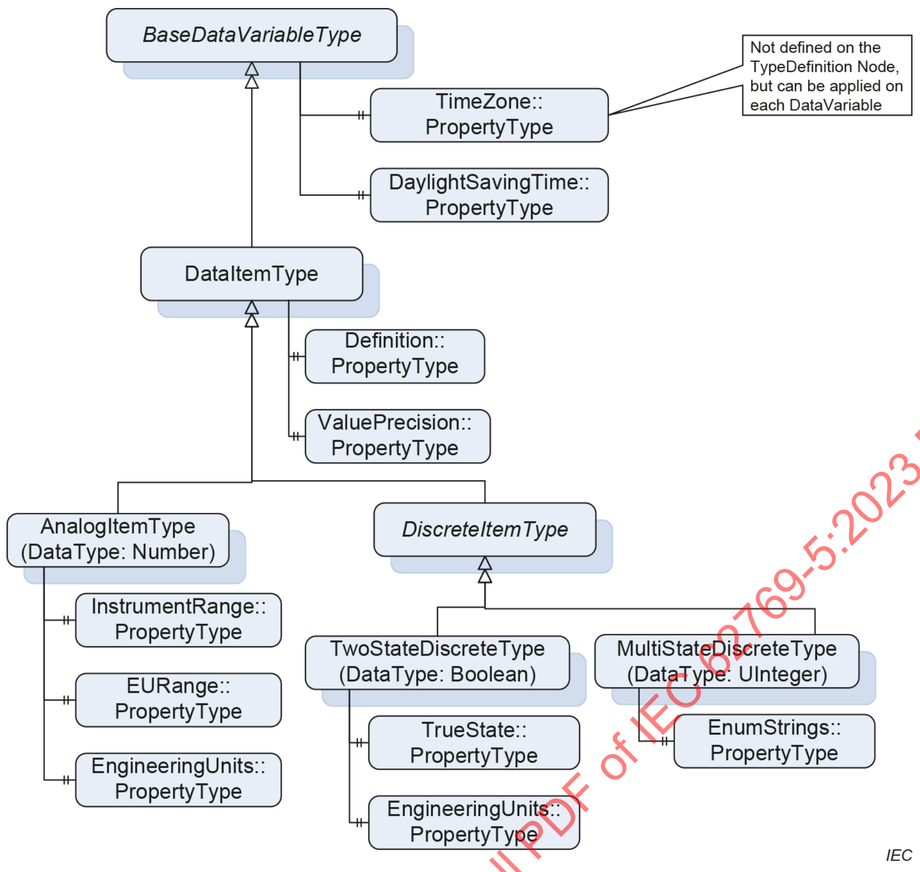

BaseDataVariableType
TimeZone:: PropertyType
DaylightSavingTime:: PropertyType
Not defined on the TypeDefinition Node, but can be applied on each DataVariable
DataItemType
Definition:: PropertyType
ValuePrecision:: PropertyType
AnalogItemType (DataType: Number)
InstrumentRange:: PropertyType
EURange:: PropertyType
EngineeringUnits:: PropertyType
DiscreteItemType
TwoStateDiscreteType (DataType: Boolean)
TrueState:: PropertyType
EngineeringUnits:: PropertyType
MultiStateDiscreteType (DataType: UInteger)
EnumStrings:: PropertyType
IEC

Figure 20 – OPC UA VariableTypes including OPC UA DataAccess

The mapping of EDD Data Types to OPC UA DataTypes and VariableTypes is specified in 15.6.

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 36.

Table 36 – 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 37.

Table 37 – 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 38.

Table 38 – 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 39.

Table 39 – Variant_Range Definition

Attributes	Value
BrowseName	Variant_Range

11 FDI® StatusCodes

11.1 General

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

11.2 Structure of the StatusCode

The general structure of the StatusCode is specified in IEC 62541-4. The exact bit assignments based on IEC 62541-4 are shown in Table 40.

Table 40 – StatusCode Bit Assignments

Field	Bit Range	Description
Severity	30 .. 31	Indicates whether the StatusCode represents a good, bad or uncertain condition. These bits have the following meanings:

Severity	Bits	Description
Good Success	00	The operation was successful; results may be used.
Uncertain Warning	01	The operation was partially successful; results might not be suitable for some purposes.
Bad Failure	10	The operation failed and any associated results cannot be used.
Reserved	11	Reserved for future use. Should also be treated as "Bad".

Field	Bit Range	Description
Reserved	29 .. 28	Reserved for future use. Shall always be zero.
SubCode	16 .. 27	The code is a numeric value assigned to represent different conditions. Each code has a symbolic name and a numeric value. All descriptions in this specification refer to the symbolic name. The numeric values are defined within the FDI® Type Libraries..
Reserved	12 .. 15	Reserved for future use. Shall always be zero.
InfoType	10 .. 11	The type of information contained in the info bits. These bits have the following meanings:

Severity	Bits	Description
NotUsed	00	The info bits are not used and shall be set to zero.
DataValue	01	The StatusCode and its info bits are associated with a data value returned from the FDI® Server.
Reserved	1X	Reserved for future use. The info bits shall be ignored.

Field	Bit Range	Description
InfoBits	0 .. 9	Additional information bits that depend on the Info Type field.

Table 41 describes the structure of the InfoBits when the Info Type is set to DataValue (01).

Table 41 – DataValue InfoBits

Info Type	Bit Range	Description
LimitBits	8 .. 9	The limit bits associated with the data value. The limits bits have the following meanings:

Limit	Bits	Description
None	00	The value is free to change.
Low	01	The value is at the lower limit for the data source.
High	10	The value is at the higher limit for the data source.
Constant	11	The value is constant and cannot change.

Info Type	Bit Range	Description
Overflow	7	If this bit is set, not every detected change has been returned since the FDI® Server’s queue buffer for the subscribed Variable reached its limit and had to purge out data.
Reserved	0 .. 6	Reserved for future use. Shall always be zero.

11.3 FDI® specific operation level result codes

IEC 62541-4 includes a set of common operational result codes which also apply to FDI®. Table 42 contains Good (success) codes that are specifically defined for FDI®.

Table 42 – 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 all the following conditions apply: – the value of the Variable has not been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Good_Edited _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 all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Good_Edited _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Good_Edited _DominantValueChanged _DependentValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value of the Variable is legal, i.e. it is within the range or a valid enumeration value; – the value of its dominant Variable was changed; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.

Table 43 contains Uncertain codes that are specifically defined for FDI®.

Table 43 – Uncertain operation level result codes

Symbolic Id	Description
Uncertain_DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when a dominant value – e.g. an engineering unit – was changed and the dependent Variable might have to be recalculated.
Uncertain_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. The "uncertain" status indicates that the dominant variable value itself has uncertain quality.
Uncertain _NoCommunicationLastUsableValue	Communication to the data source has failed. The Variable value is the last value that had a good quality.
Uncertain_LastUsableValue	Whatever was updating this value will no longer be doing so.
Uncertain_SubstituteValue	The value is an operational value that was manually overwritten.
Uncertain_InitialValue	The value is an initial value for a Variable that normally receives its value from another Variable.
Uncertain_SensorNotAccurate	The value is at one of the sensor limits.
Uncertain_EngineeringUnitsExceeded	The value is outside of the range of values defined for this parameter.
Uncertain_SubNormal	The value is derived from multiple sources and has less than the required number of good sources.

Table 44 contains Bad codes that are specifically defined for FDI®.

Table 44 – Bad operation level result codes

Symbolic Id	Description
Bad_Edited_OutOfRange	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a Variable value when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value).
Bad_InitialValue_OutOfRange	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a Variable value when all the following conditions apply: – the value is an initial value for a Variable that normally receives its value from another Variable; – the value is not a legal value (e.g. out of range or an invalid enumeration value).
Bad_DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable value when a dominant value – e.g. an engineering unit – was changed and the dependent Variable cannot be accessed.
Bad_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.
Bad_OutOfRange _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.

Symbolic Id	Description
Bad_Edited _OutOfRange _DominantValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Bad_OutOfRange _DominantValueChanged _DependenValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – all the changes have not been applied, yet.
Bad_Edited _OutOfRange _DominantValueChanged _DependenValueChanged	This status applies to Variable values that are part of an EditContext (see 9.3). It is delivered with a dependent Variable when all the following conditions apply: – the value of the Variable has been changed; – the value is not a legal value (e.g. out of range or an invalid enumeration value); – the value of its dominant Variable was changed; – the value of any of its dependent Variables was changed; – all the changes have not been applied, yet.
Bad_ConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
Bad_DeviceFailure	There has been a failure in the device/data source that generates the value.
Bad_NodeInvalid	The identifier does not refer to a valid Node in the Device Model. This result code is used both as service- and as operation-level result code.
Bad_NotConnected	The Variable should receive its value from another Variable, but has never been configured to do so.
Bad_OutOfService	The source of the data is not operational.
Bad_SensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source.
Bad_UIPHandleInvalid	The handle does not refer to a subscribed Node Attribute.
Bad_WaitingForInitialData	Waiting for the FDI® Server to obtain values from the underlying data source. After subscribing to Variables, it might take some time before values are delivered. In such cases, an initial update might be sent with this status prior to the Notification with the first valid value.

12 Specialized topology elements

Devices and other topology elements can be specialized using the modelling elements defined in this document 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 45 and Table 46 specify the arguments and AddressSpace representation, respectively.

Signature

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

Table 45 – LogAuditTrailMessage Method Arguments

Argument	Description
Message	Free text describing the context.

Table 46 – 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 47.

Table 47 – 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 can 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 can 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 48 specifies the mapping of Properties to Package Catalog elements.

Table 48 – 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 15.6 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 15.8 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 Modular Device

Subclause 15.4 specifies the mapping of a modular device's package.

As described in IEC 62769-4, a package for a modular device contains a head station EDD and one or more module EDDs. The head station EDD contains COMPONENT constructs that identify the module EDDs.

The EDDs for head station Device and for each module shall be individually mapped according to the mapping rules for single Devices.

The head station Device will show up as a modular Device as specified in IEC 62541-100.

- The SubDevices component in the instance of the head Device references instantiated modules.
- The SupportedTypes folder in the SubDevices component references all DeviceTypes for modules that can appear in the instance of the head Device.

15.5 Block

15.5.1 General

Subclause 15.5 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 can appear in a Device Instance. The Blocks component in a Device Instance references instantiated Blocks.

15.5.2 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.

15.5.3 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 exists on the Type, the offline and the online instances.

See 15.6 on how EDDL attributes are used to create Parameter nodes.

15.5.4 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.

15.5.5 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 15.8 on how EDDL attributes are used to create Action Nodes.

15.5.6 Mapping to MethodSet

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

15.5.7 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.

15.6 Parameter

15.6.1 General

EDDL Parameters (for devices or blocks) are mapped to OPC UA Variables. The VariableType used can 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 document includes additional VariableTypes for more sophisticated EDDL types. See Table 50 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 49 summarizes the Variable Attributes and describes how they are set from the EDDL description information.

Table 49 – Setting OPC UA Variable Attributes from EDDL variable attributes

Attributes	Description
Value	The most recent value of the Variable that the FDI® Server has read from the device.
DataType	The EDDL data type is translated into an OPC UA standard data type according to Table 50.
ValueRank	Either set to "Scalar" or – when the Parameter is an array – the number of dimensions.
ArrayDimensions	Specifies the length of each dimension if the Parameter is an array. This attribute is not exposed if the length is unknown or dynamic.
AccessLevel	The AccessLevel Attribute is set based on the EDDL variable HANDLING attribute according to the following table: **Field** – **Bit** – **Description** CurrentRead – 0 – Set if EDDL variable HANDLING is defined as READ. Reset otherwise. CurrentWrite – 1 – Set if EDDL variable HANDLING is defined as WRITE. Reset otherwise. If the HANDLING attribute is missing, the Parameter will be defined as readable and writeable.
UserAccessLevel	The AccessLevel with possible restrictions based on client identity as defined by the FDI® Server.
MinimumSamplingInterval	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).

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 50 shows the correspondence between the EDDL types and sizes and the OPC UA VariableTypes and standard DataTypes.

Table 50 – 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 15.6.5	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 15.6.6	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 intervals since January 1, 1601 (UTC))	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 service returns an appropriate error in the serviceResult and the diagnosticInfo data members of the ResponseHeader and does not accept the data.

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
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	TIME_VALUE is the "number of 1/32 ms". The "Duration" is calculated by multiplying the TIME_VALUE with 0,031 25 (which is the float equivalent for 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.

EDDL Data Type	OPC UA VariableType	OPC UA DataType	Constraints
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 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 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.
OCTET	BaseDataVariableType	ByteString	
INDEX	See 15.6.5	String	
BOOLEAN	BaseDataVariableType	Boolean	

As Table 50 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. Subclause 15.6 is intended to provide details for each EDDL data type.

15.6.2 Private Parameters

The Parameters specified in an FDI® Package can be declared private using the PRIVATE Attribute specified in IEC 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.

15.6.3 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.

15.6.4 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 affected variables using that unit code shall be refreshed. When an affected 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.

15.6.5 Enumerated Parameters

If no semantic map has been applied to an enumerated variable, an OPC UA DataVariable with the VariableType MultiStateValueDiscreteType (defined in IEC 62541-8) is used for each enumerator in the EDDL enumerated variable definition – otherwise the VariableType MultiStateDictionaryEntryDiscreteType shall be used.

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.

15.6.6 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.

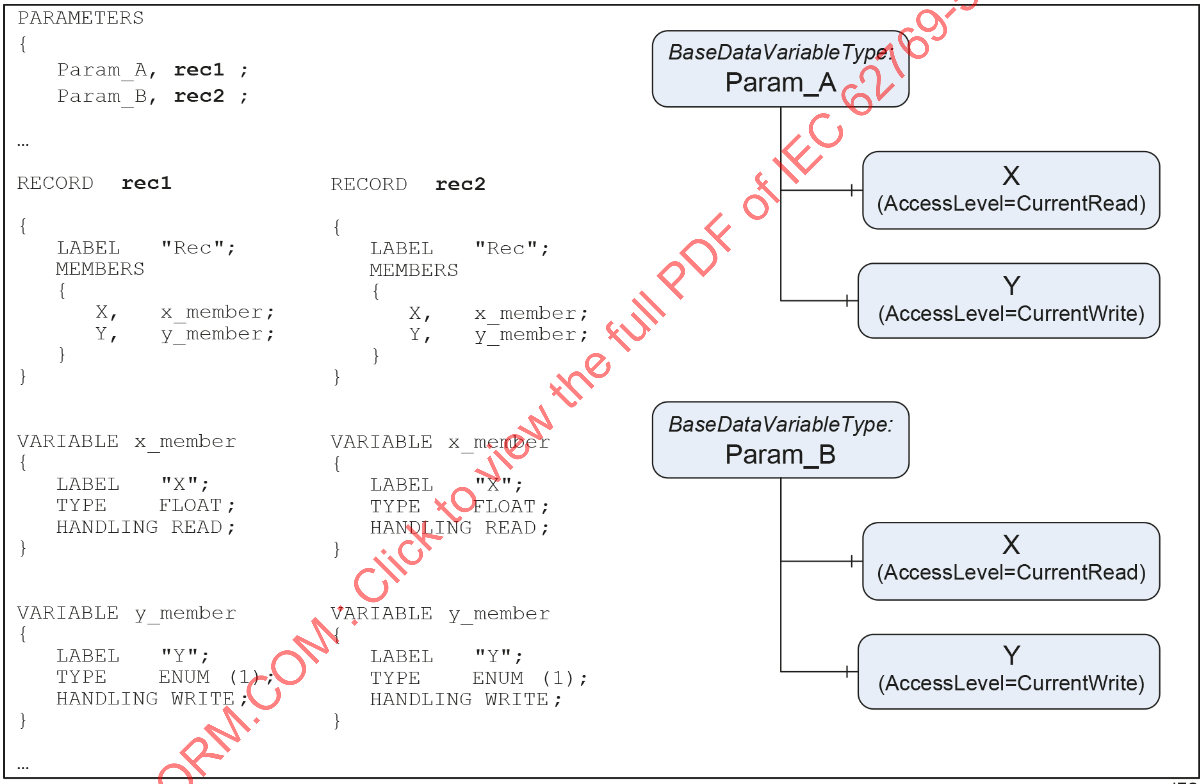
The Boolean array of the DataVariable reflects the status of all bits. TRUE is used when a bit is set and FALSE when a bit is reset.

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.

15.6.7 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 21 for an example of how records are represented in the OPC UA AddressSpace.



```
PARAMETERS
{
   Param_A, rec1 ;
   Param_B, rec2 ;

…

RECORD   rec1

{
   LABEL    "Rec";
   MEMBERS
   {
      X,    x_member;
      Y,    y_member;
   }
}


VARIABLE x_member
{
   LABEL    "X";
   TYPE     FLOAT;
   HANDLING READ;
}


VARIABLE y_member
{
   LABEL    "Y";
   TYPE     ENUM (1);
   HANDLING WRITE;
}

…
```

```
RECORD   rec2

{
   LABEL    "Rec";
   MEMBERS
   {
      X,    x_member;
      Y,    y_member;
   }
}


VARIABLE x_member
{
   LABEL    "X";
   TYPE     FLOAT;
   HANDLING READ;
}


VARIABLE y_member
{
   LABEL    "Y";
   TYPE     ENUM (1);
   HANDLING WRITE;
}
```

BaseDataVariableType: Param_A
X (AccessLevel=CurrentRead)
Y (AccessLevel=CurrentWrite)
BaseDataVariableType: Param_B
X (AccessLevel=CurrentRead)
Y (AccessLevel=CurrentWrite)
IEC

Figure 21 – 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 21, 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.

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

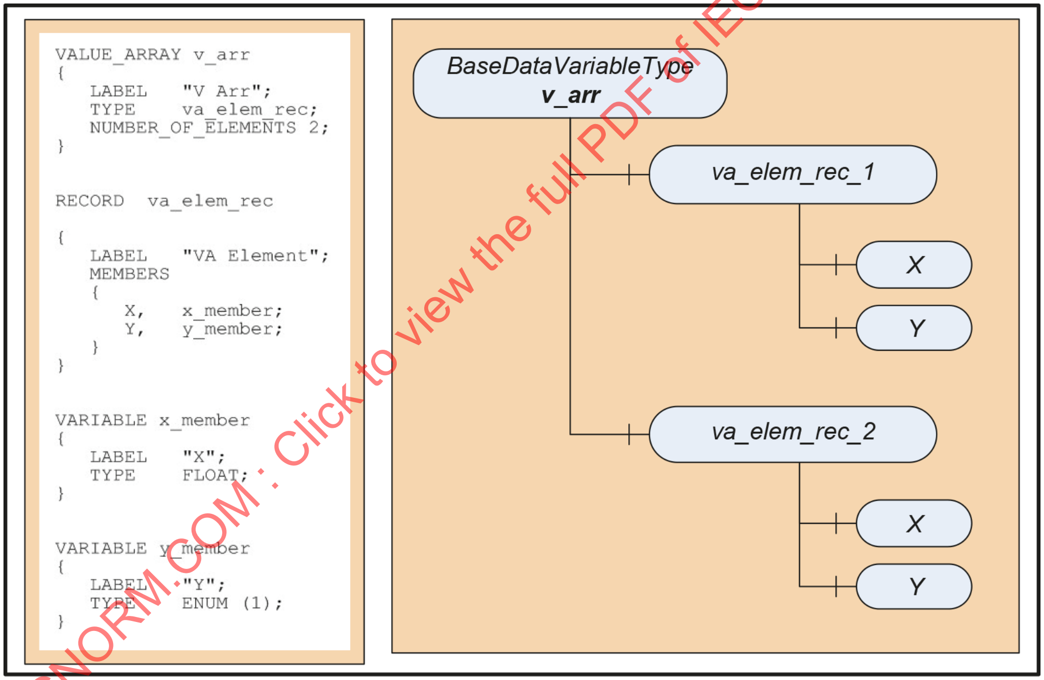
A single DataVariable will represent an EDDL VALUE_ARRAY or LIST item when the data type of the referenced array element has a simple data type.

The OPC UA DataVariable Attributes are set as follows:

- DataType is set to the type of the array element (see Table 50 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 ArrayDimensions Attribute. In the case of an EDDL LIST, the ArrayDimensions Attribute is not exposed. The number of elements is unspecific since the size can change dynamically.

15.6.9 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 22 shows the EDDL sample code of a VALUE_ARRAY of RECORDs and the corresponding complex variable in the OPC UA AddressSpace.



```
VALUE_ARRAY v_arr
{
    LABEL    "V Arr";
    TYPE     va_elem_rec;
    NUMBER_OF_ELEMENTS 2;
}

RECORD  va_elem_rec

{
    LABEL    "VA Element";
    MEMBERS
    {
        X,    x_member;
        Y,    y_member;
    }
}

VARIABLE x_member
{
    LABEL    "X";
    TYPE     FLOAT;
}

VARIABLE y_member
{
    LABEL    "Y";
    TYPE     ENUM (1);
}
```

Figure 22 – 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.

15.6.10 Representation of COLLECTION and REFERENCE ARRAY

The EDDL constructs COLLECTION and REFERENCE ARRAY are mapped to Functional Group Objects (see 15.7) when used for grouping only.

15.6.11 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.

15.6.12 EDDL CLASS Attributes on Parameters

The EddlDictionary Object is a dictionary with the purpose of representing EDDL CLASS Attributes for Parameters. "HasDictionaryEntry" references are used to associate a parameter with EDDL CLASS attributes.

Note that only these EDDL CLASS attributes are mapped to the FDI® Information Model, that are used in a consistent way by EDDL implementations.

The OPC UA types and objects are defined in Table 51, Table 52, and Table 53.

Table 51 – Definition of EddlDictionaryType

Attribute	Value				
BrowseName	**EddlDictionaryEntryType**				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
SubType of DictionaryEntry defined in OPC UA for Devices (DI) Part 100					

Table 52 – Definition of EddlDictionary Object

Attribute	Value				
BrowseName	EddlDictionary				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Component of the DataDictionaries Object defined in IEC 62541-100					
HasTypeDefinition	ObjectType	DictionaryFolderType			
HasComponent	Object	ParameterClass		DictionaryFolderType	

Table 53 – Definition of Parameter Class Attributes

Attribute	Value				
BrowseName	**ParameterClass**				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasTypeDefinition	ObjectType	DictionaryFolderType			
HasComponent	Object	Alarm		EddlDictionaryEntryType	
HasComponent	Object	Contained		EddlDictionaryEntryType	
HasComponent	Object	Correction		EddlDictionaryEntryType	
HasComponent	Object	Device		EddlDictionaryEntryType	
HasComponent	Object	Diagnostic		EddlDictionaryEntryType	
HasComponent	Object	Dynamic		EddlDictionaryEntryType	
HasComponent	Object	Input		EddlDictionaryEntryType	
HasComponent	Object	Local		EddlDictionaryEntryType	
HasComponent	Object	Operate		EddlDictionaryEntryType	
HasComponent	Object	Output		EddlDictionaryEntryType	
HasComponent	Object	Service		EddlDictionaryEntryType	
HasComponent	Object	Specialist		EddlDictionaryEntryType	
HasComponent	Object	Temporary		EddlDictionaryEntryType	
HasComponent	Object	Tune		EddlDictionaryEntryType	

To illustrate the model, an example is shown in Figure 23.

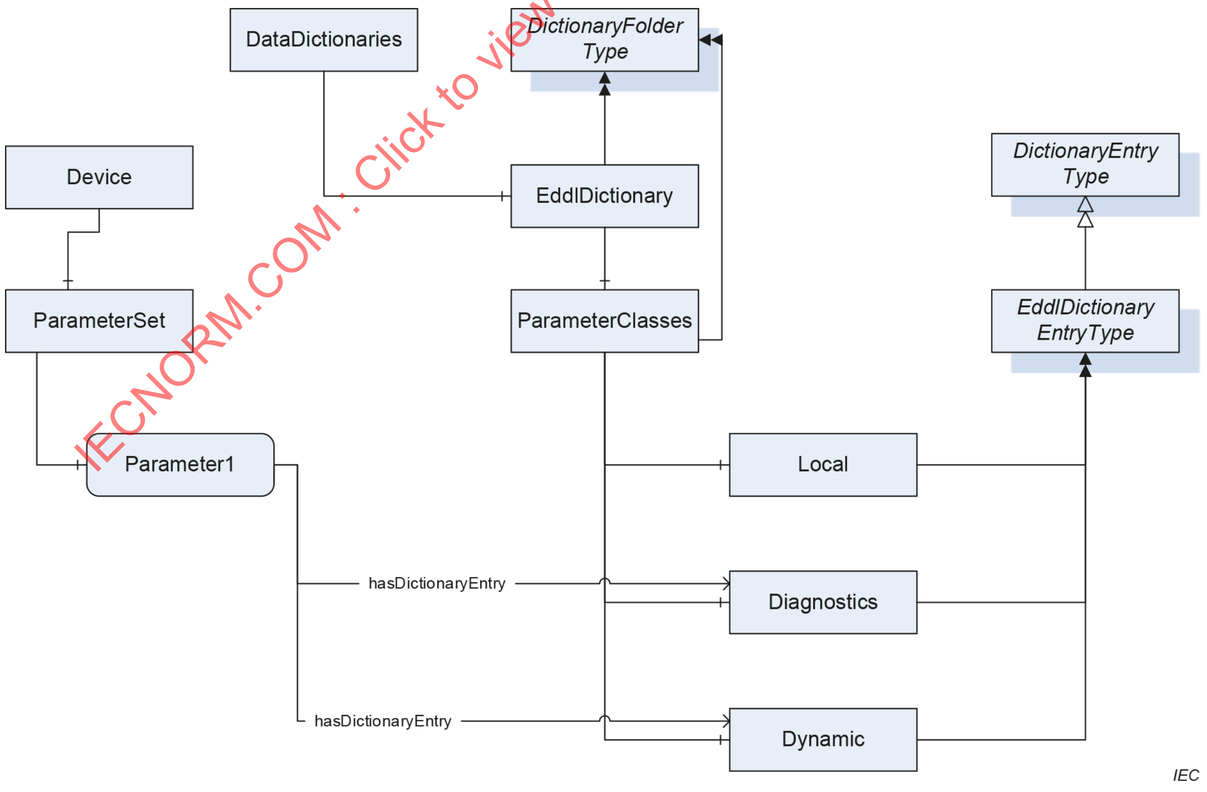


Figure 23 – Example of EDDL CLASS Attributes in the FDI® OPC UA Information Model

15.7 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 building 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.

15.8 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.

15.9 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 can be declared private using the PRIVATE Attribute specified in IEC 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.

15.10 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.

15.11 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.

15.12 Semantic Identifies

Semantic identifiers based on the identification schema ISO/IEC 27005 are mapped to EDDL constructs (e.g. COLLECTION, VARIABLE) by the EDDL construct SEMANTIC_MAP (see IEC 61804-3). In the OPC-UA address space, the semantic identifiers are represented by concrete object types derived from the abstract type DictionaryEntryType. The concrete reference type HasDictionaryEntry binds the semantic object to the OPC-UA object representing the EDDL construct (e.g. COLLECTION, VARIABLE). SEMANTIC_MAP can also be used to map enumeration items to semantic identifiers (e.g. units).

The EDDL SEMANTIC_MAP construct maps a list (one or more) keys to a list (one or more) values. The keys are data dictionary ids and the values are EDDL item identifier.

A DictionaryEntryType Object is created for each key term (data dictionary id) of the EDDL SEMANTIC_MAP. The key value is mapped to the BrowseName and NodeId property of the DictionaryEntryType Object.

A HasDictionaryEntry reference is created for each value listed in the EDDL SEMANTIC_MAP construct with the SourceNode set to the NodeId of the OPC UA object representing EDDL item identified.

15.13 DictionaryIds Property

For enumerated Variables or Enumeration DataTypes (see IEC 61804-3), the enumeration values may have a specific reference to a dictionary entry. To express this, Enumeration and OptionSet DataTypes or Variables with VariableTypes like MultiStateDiscreteType, MultiStateValueDiscreteType or OptionSetType reference an additional Property with the BrowseName DictionaryIds which provides a Value with an array of DataType NodeId where each array item contains the NodeId of the corresponding DictionaryEntryType Object. The DictionaryIds Property is formally defined in Table 54.

Table 54 – DictionaryIds Definition

Attribute	Value
BrowseName	DictionaryIds
IsAbstract	False
TypeDefinition	PropertyType
DataType	NodeId[]

15.14 MultiStateDictionaryEntryDiscreteType

This VariableType defines the general characteristics of a DiscreteItem that can have more than two states including semantic definitions. The MultiStateDictionaryEntryDiscreteType derives from the MultiStateDiscreteType. It is formally defined in Table 55.

Table 55 – MultiStateDictionaryEntryDiscreteType definition

Attribute	Value				
BrowseName	MultiStateDictionaryEntryDiscreteType				
NodeClass	VariableType				
ValueRank	Scalar				
DataType	UInteger				
IsAbstract	FALSE				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the MultiStateDiscreteType defined in IEC 62541-8; i.e the Properties of that type are inherited.					
HasProperty	Variable	EnumDictionary Entries	NodeId[][]	PropertyType	Optional
HasProperty	Variable	ValueAsDictionary Entries	NodeId[]	PropertyType	Mandatory

EnumDictionaryEntries is a two-dimensional array of NodeIds. The first dimension is used to list all possible dictionary entry values for the related variable in a specific dictionary (e.g. CDD or eCl@ss). The second dimension is used to reference this dictionary. The size of the first array dimension shall be the same as the size of EnumStrings.

ValueAsDictionaryEntries provides a list of all dictionary entry values in the different dictionaries related to the current value of the variable.

If an instance of this type is writeable, the property ValueAsDictionaryEntries shall be writeable as well. Clients are allowed to write an array with only one entry and the server is responsible to resolve additional appropriate entries. This will have the same result as writing the value attribute, but the client does not need knowledge about the numeric values.

The NodeIds represent the dictionary entries and can be generated with dictionary knowledge.

15.15 GetNodeIdsByDictionaryEntryId

GetNodeIdsBySemanticId is a custom query used to get the NodeIds for nodes that have a HasDictionaryEntry Reference to a DictionaryEntryType Object matching the provided DictionaryEntryId. The Method returns only nodes that are referenced by the StartingNode with the defined ReferenceType.

Signature

```
   GetNodeIdsByDictionaryEntryId (
       [in] NodeId StartingNode
       [in] NodeId ReferenceType
       [in] NodeId DictionaryEntryId
       [out] NodeId[] Nodes
     );
```

The Method arguments and result codes are described in Table 56 and Table 57.

Table 56 – GetNodeIdsByDictionaryEntryId Method arguments

Argument	Description
StartingNode	A NodeId defining the starting node for the query.
ReferenceType	A NodeId defining the ReferenceType to take into account for the query.
DictionaryEntryId	A NodeId defining the DictionaryEntry Object that represents the queried dictionary entry.
Nodes	Contains an Array of NodeIds of the nodes that match the provided query parameters.

Table 57 – GetNodeIdsByDictionaryEntryId Method result codes

Result Code	Description
Bad_InvalidArgument	See IEC 62541-4 for a general description. Invalid file handle in call.
Bad_NotWritable	See IEC 62541-4 for a general description. File might be locked and thus not writable.
Bad_InvalidState	See IEC 62541-4 for a general description. File was not opened for write access.

Table 58 specifies the AddressSpace representation for the GetNodeIdsByDictionaryEntryId Method.

Table 58 – GetNodeIdsByDictionaryEntryId

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

16 Profiles

This chapter defines the profiles and conformance units for the OPC UA Information Model for FDI® Clients and FDI® Servers. Profiles are named groupings of conformance units. Facets are profiles that will be combined with other Profiles to define the complete functionality of an OPC UA Server or Client.

FDI® Servers (FDI® Hosts) that implement a subset of the FDI® Information Model that is relevant for generic OPC UA Clients shall implement as a minimum the BaseDevice_Server_Facet specified in IEC 62541-100 (OPC UA for Devices).

Table 59 specifies the facet for FDI® Servers (FDI® Hosts) that support FDI® Clients.

Table 59 – FDI® Server Facet Definition

Conformance Unit	Description	Optional/ Mandatory
FDI® Information Model	Support Objects that conform to the Types specified in this document and in IEC 62541-100.	M

Table 60 defines the facet for FDI® Clients that use the FDI® Information Model.

Table 60 – FDI® Client Facet Definition

Conformance Unit	Description	Optional/ Mandatory
FDI® Client Information Model	Consume Objects that conform to the Types specified in this document and in IEC 62541-100.	M

Annex A
(normative)

Namespace and Mappings

This annex defines the numeric identifiers for all of the numeric NodeIds defined in this document. 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.

An electronic version of the complete Information Model defined in this document 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 is in a separate file with name:

OPCUA_Part5_Model_V1.1\Opc.Ua.FDI5.NodeSet2.xml

Bibliography

IEC 61987 (all parts), *Industrial-process measurement and control – Data structures and elements in process equipment catalogues*

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

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

ISO/IEC 27005, *Information security, cybersecurity and privacy protection − Guidance on managing information security risks*

NAMUR NE107, *Self-Monitoring and Diagnosis of Field Devices*
<available at www.namur.de > [viewed 2023-02-02]

____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

____

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 5: Modèle d'Information FDI®

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets.

L'IEC 62769-5 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels. Il s'agit d'une Norme internationale.

Cette troisième édition annule et remplace la deuxième édition parue en 2021. Cette édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

a) ajout d'INTERACTIVE_TRANSFER_TO_DEVICE ACTION.

Le texte de cette Norme internationale est issu des documents suivants:

Projet	Rapport de vote
65E/858/CDV	65E/915/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à son approbation.

La langue employée pour l'élaboration de cette Norme internationale est l'anglais.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2, il a été développé selon les Directives ISO/IEC, Partie 1 et les Directives ISO/IEC, Supplément IEC, disponibles sous www.iec.ch/members_experts/refdocs. Les principaux types de documents développés par l'IEC sont décrits plus en détail sous www.iec.ch/publications.

Une liste de toutes les parties de la série IEC 62769, publiées sous le titre général *Intégration des appareils de terrain (FDI®)*, se trouve sur le site web de l'IEC.

Le comité a décidé que le contenu de ce document ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous webstore.iec.ch dans les données relatives au document recherché. A cette date, le document sera

- reconduit,
- supprimé,
- remplacé par une édition révisée, ou
- amendé.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de ce document indique qu'il contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer ce document en utilisant une imprimante couleur.

INTÉGRATION DES APPAREILS DE TERRAIN (FDI®) –

Partie 5: Modèle d'Information FDI®

1 Domaine d'application

La présente partie de l'IEC 62769 définit le Modèle d'Information FDI®[1]. L'un des principaux objectifs du Modèle d'Information est de refléter la topologie du système d'automatisation. Par conséquent, il représente les appareils du système d'automatisation ainsi que les réseaux de communication connectés, y compris leurs propriétés, leurs relations et les opérations dont ils peuvent faire l'objet. Les types présents dans l'AddressSpace du Serveur FDI® constituent un catalogue, qui est créé à partir des Paquetages FDI®.

Les types fondamentaux pour le Modèle d'Information FDI® sont définis dans l'OPC UA pour les Appareils (IEC 62541-100). Le Modèle d'Information FDI® spécifie des extensions pour quelques cas spéciaux et explique la façon dont ces types sont utilisés et dont les contenus sont construits à partir des éléments de DevicePackages.

L'architecture FDI® complète est représentée à la Figure 1. Les composants architecturaux qui relèvent du domaine d'application du présent document ont été mis en évidence dans cette représentation.

[1] FDI® est une marque déposée de l'organisation à but non lucratif Fieldbus Foundation, Inc. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve le détenteur de la marque ou l'emploi de ses produits. La conformité n'exige pas l'utilisation de la marque. L'utilisation de la marque exige l'autorisation du détenteur de la marque.

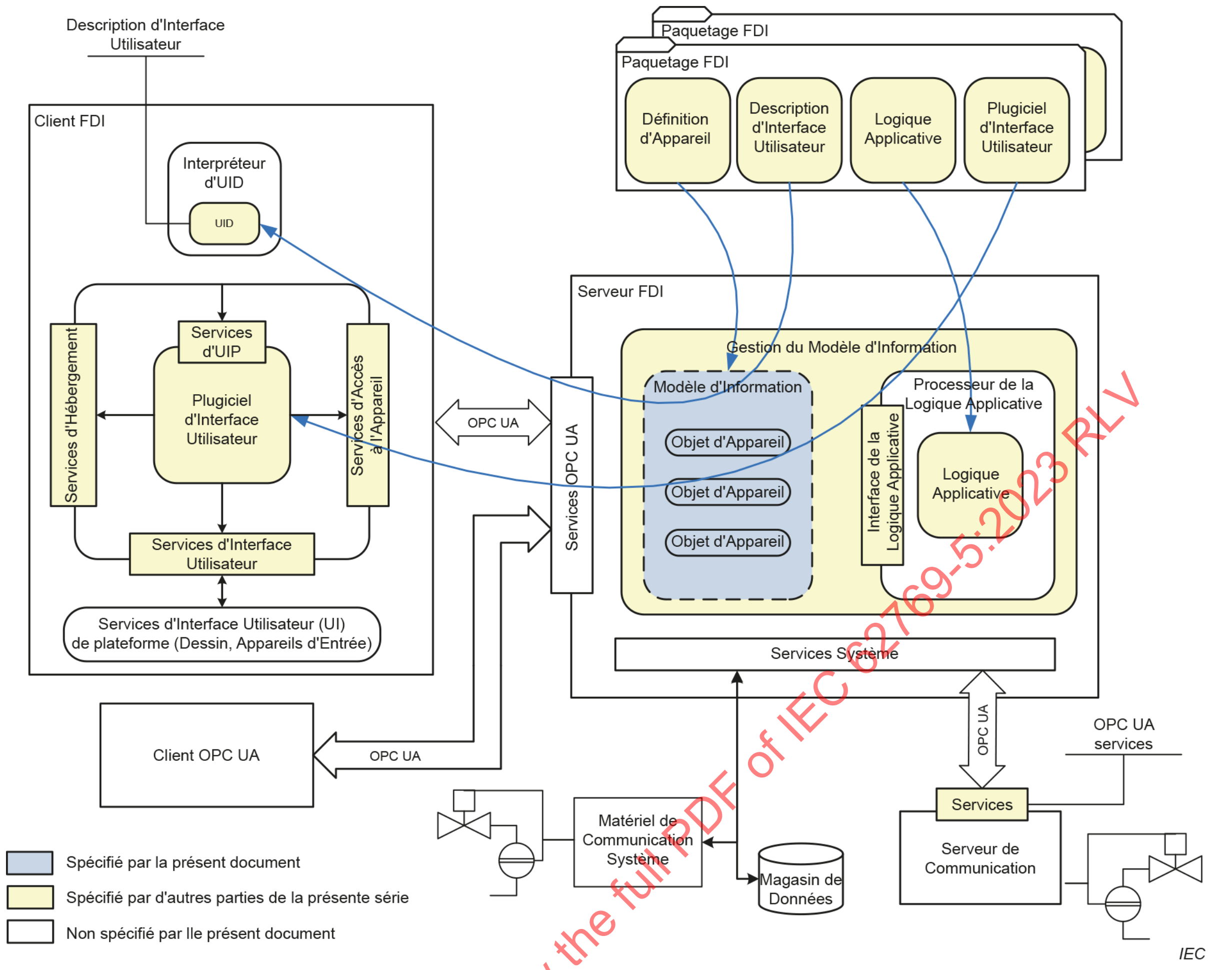

Description d'Interface Utilisateur
Client FDI
Interpréteur d'UID
UID
Services d'UIP
Plugiciel d'Interface Utilisateur
Services d'Hébergement
Services d'Accès à l'Appareil
Services d'Interface Utilisateur
Services d'Interface Utilisateur (UI) de plateforme (Dessin, Appareils d'Entrée)
Paquetage FDI
Paquetage FDI
Définition d'Appareil
Description d'Interface Utilisateur
Logique Applicative
Plugiciel d'Interface Utilisateur
OPC UA
Services OPC UA
Serveur FDI
Gestion du Modèle d'Information
Modèle d'Information
Objet d'Appareil
Objet d'Appareil
Objet d'Appareil
Interface de la Logique Applicative
Processeur de la Logique Applicative
Logique Applicative
Services Système
Client OPC UA
OPC UA
OPC UA
OPC UA services
Services
Serveur de Communication
Matériel de Communication Système
Magasin de Données
Spécifié par la présent document
Spécifié par d'autres parties de la présente série
Non spécifié par lle présent document
IEC

Figure 1 – Diagramme de l'architecture FDI®

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61784-1-3:2023, *Réseaux industriels – Profils – Partie 1-3: Profils de bus de terrain – Famille de profils de communication 3*

IEC 61804-3, *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produits (EDDL) – Partie 3: Sémantique et syntaxe EDDL*

IEC 61804-4, *Les dispositifs et leur intégration dans les systèmes de l'entreprise – Blocs fonctionnels (FB) pour les procédés industriels et le langage de description électronique de produits (EDDL) – Partie 4: Interprétation EDD*

IEC 62541-3, *Architecture unifiée OPC – Partie 3: Modèle d'espace d'adressage*

IEC 62541-4, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'information*

IEC 62541-6, *Architecture unifiée OPC – Partie 6: Mappings*

IEC 62541-8, *Architecture unifiée OPC – Partie 8: Accès aux données*

IEC 62541-100, *Architecture unifiée OPC – Partie 100: Interface d'appareils*

IEC 62769-1, *Intégration des appareils de terrain (FDI®) – Partie 1: Vue d'ensemble*

IEC 62769-2, *Intégration des appareils de terrain (FDI®) – Partie 2: Client*

IEC 62769-3, *Intégration des appareils de terrain (FDI®) – Partie 3: Serveur*

IEC 62769-4, *Intégration des appareils de terrain (FDI®) – Partie 4: Paquetages FDI®*

IEC 62769-6, *Intégration des appareils de terrain (FDI®) – Partie 6: Mappings de technologies FDI®*

IEC 62769-7, *Intégration des appareils de terrain (FDI®) – Partie 7: Appareils de communication*

IEC 62769-1xx (toutes les parties), *Intégration des appareils de terrain (FDI®) – Partie 1xx-y: Profils*

OPC 10000-19, *OPC Unified Architecture – Part 19: Dictionary Reference* (disponible an anglaise seulement

3 Termes, définitions, abréviations, acronymes et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions de l'IEC 62769-1 et de l'IEC 62769-3 s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse https://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse https://www.iso.org/obp

3.2 Abréviations et acronymes

Pour les besoins du présent document, les abréviations et acronymes de l'IEC 62769-1 ainsi que les suivants s'appliquent.

IHM	Interface homme-machine
SCADA (Supervisory Control and Data Acquisition)	Commande de surveillance et d'acquisition de données
TCP (Transmission Control Protocol)	Protocole de contrôle de transmission

3.3 Conventions

3.3.1 Utilisation de majuscules

La mise en majuscules de la première lettre des mots est utilisée dans la série IEC 62769 pour souligner un terme défini spécifique à la FDI®.

3.3.2 Conventions pour la notation graphique

L'OPC UA définit une notation graphique pour un AdressSpace de l'OPC UA. Elle définit des symboles graphiques pour l'ensemble des NodeClasses et concernant la façon dont les différents types de Références entre les Nœuds peuvent être visualisés. La Figure 2 représente les symboles pour les NodeClasses utilisés dans le présent document. Les types qui représentent des NodeClasses comportent toujours une ombre.

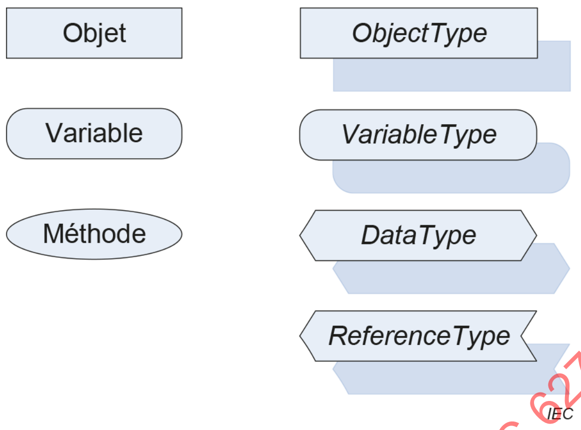


Figure 2 – Notation graphique de l'OPC UA pour les NodeClasses

La Figure 3 représente les symboles pour les ReferenceTypes utilisés dans le présent document. Le symbole de Référence pointe généralement du Nœud d'origine vers le Nœud cible. La seule exception est la Référence HasSubType. Les Références les plus importantes, telles que HasComponent, HasProperty, HasTypeDefinition et HasSubType comportent des symboles spécifiques qui permettent de ne pas indiquer le nom de la Référence. Pour les autres ReferenceTypes ou ReferenceTypes dérivés, le nom du ReferenceType est utilisé avec le symbole.

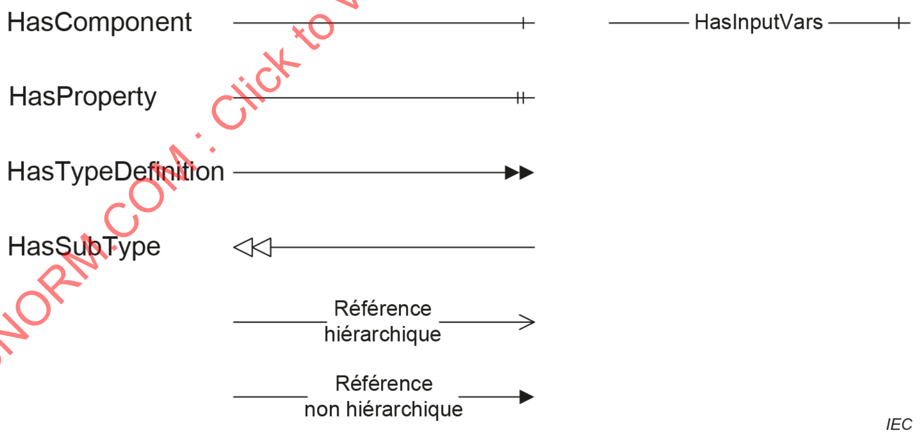


Figure 3 – Notation graphique de l'OPC UA pour les Références

La Figure 4 représente un exemple type de l'utilisation de la notation graphique. Object_A et Object_B sont des instances d'ObjectType_Y indiquées par les Références HasTypeDefinition. L'ObjectType_Y est dérivé de l'ObjectType_X indiqué par la Référence HasSubType. L'Object_A comporte les composants Variable_1, Variable_2 et Method_1.

Afin de décrire les composants d'un Object sur l'ObjectType, les mêmes NodeClasses et Références sont utilisées sur l'Object et sur l'ObjectType, comme cela est indiqué dans l'exemple pour l'ObjectType_Y. Les Nodes utilisés pour décrire un ObjectType sont des Nodes de déclaration d'instance.

Afin de fournir des informations plus détaillées pour un Node, un sous-ensemble ou l'ensemble des Attributs et leurs valeurs peuvent être ajoutés à un symbole graphique (voir, par exemple, Variable_1, le composant de l'Object_A à la Figure 4).

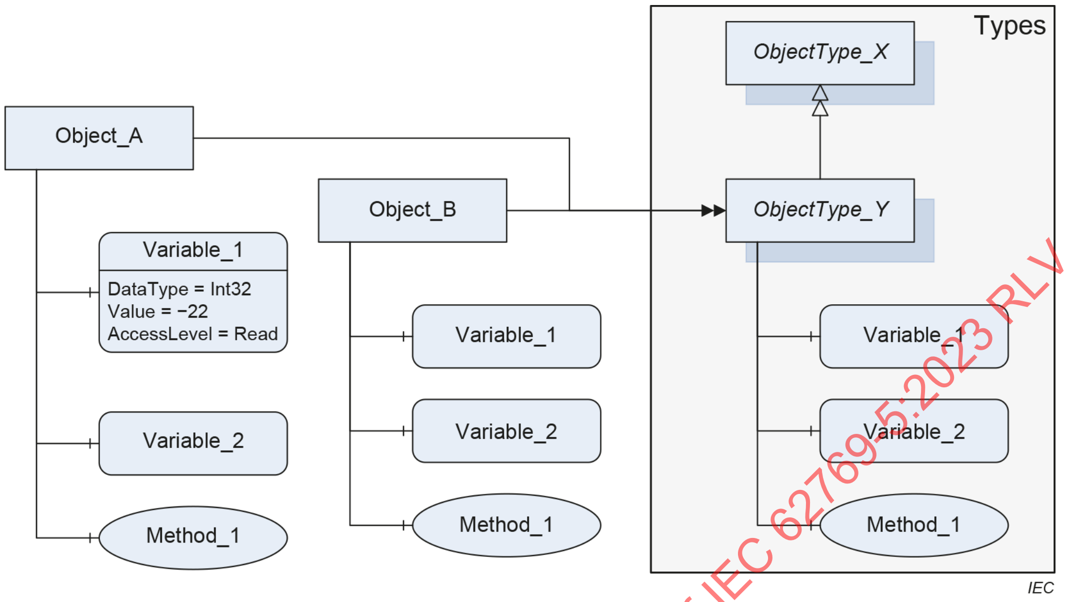


Figure 4 – Exemple de notation graphique de l'OPC UA

Afin d'améliorer la lisibilité, le présent document inclut fréquemment le nom de type à l'intérieur de la boîte d'instance, plutôt que d'afficher les deux boîtes et la référence entre celles-ci. Cette optimisation est représentée à la Figure 5.

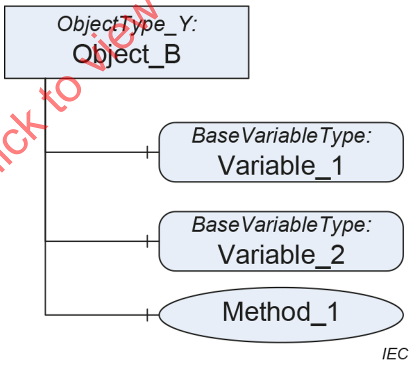


Figure 5 – Référence de Type optimisée

4 Vue d'ensemble de l'Architecture Unifiée OPC

4.1 Généralités

Le principal cas d'utilisation pour les normes OPC est l'échange de données en ligne entre des appareils et des systèmes IHM ou SCADA. Dans ce cas d'utilisation, les données de l'appareil sont fournies par un serveur OPC et sont consommées par un client OPC intégré au système IHM ou SCADA. L'OPC fournit une fonctionnalité qui permet de naviguer dans des espaces de nom hiérarchiques qui contiennent des éléments de données afin de lire, d'écrire et de surveiller ces éléments pour les modifications de données.

L'OCP UA comprend des fonctionnalités telles que l'Accès aux Données, les Alarmes et les Données Historiques, par l'intermédiaire de mécanismes de communication indépendants de la plateforme et des capacités de modélisation génériques, extensibles et orientées objet, pour les informations qu'un système souhaite exposer.

La version actuelle de l'OPC UA définit un protocole TCP binaire optimisé pour la communication intranet haute performance ainsi qu'un mapping aux Services web. Le modèle de service abstrait ne dépend pas d'un mapping à un protocole spécifique et permet d'ajouter de nouveaux protocoles ultérieurement. Des fonctionnalités telles que la sécurité, le contrôle de l'accès et la fiabilité sont directement incluses dans les mécanismes de transport. Selon l'indépendance des protocoles vis-à-vis de la plateforme, les serveurs et les clients OPC UA peuvent être directement intégrés dans les appareils et les contrôleurs.

Le modèle d'information OPC UA apporte aux serveurs une méthode normalisée pour exposer des Objets aux Clients. Dans les termes de l'OPC UA, les Objets sont composés d'autres Objets, de Variables et de Méthodes. L'OPC UA permet également d'exprimer des relations avec d'autres Objets.

L'ensemble des Objets et des informations connexes qu'un Serveur OPC UA met à la disposition des Clients est appelé son AdressSpace. Les éléments d'un Modèle d'Objet OPC UA sont représentés dans l'AdressSpace par un ensemble de Nœuds décrits par des Attributs et interconnectés par des Références. L'OPC UA définit différentes classes de Nœuds pour représenter les composants de l'AddressSpace, notamment les Objets (Objects), Variables, Méthodes (Methods), ObjectTypes (Types d'Objets), DataTypes (Types de Données) et ReferenceTypes (Types de Références). Chaque NodeClass possède un ensemble défini d'Attributs.

Les Objets (Objects) sont utilisés pour représenter des composants tels que des dossiers, des Appareils ou des Réseaux. Un Objet est associé à un ObjectType correspondant qui fournit les définitions de cet Objet.

Les Variables sont utilisées pour représenter des valeurs. Deux catégories de Variables sont définies, les Propriétés et les DataVariables.

Les Propriétés (Properties) sont des caractéristiques définies par un Serveur pour les Objets, les DataVariables et d'autres Nœuds. Il n'est pas admis de définir des Propriétés pour d'autres Propriétés. La Propriété Manufacturer (Fabricant) d'un Appareil est un exemple de Propriété d'un Objet.

Les DataVariables représentent le contenu d'un Objet. Les DataVariables peuvent avoir des DataVariables comme composants. Elles sont utilisées généralement par les Serveurs pour exposer des éléments individuels de matrices et de structures. Le présent document utilise les DataVariables principalement pour représenter les Paramètres des Appareils.

4.2 Vue d'ensemble des Appareils OPC UA

La norme Architecture unifiée OPC pour Appareils (DI) (IEC 62541-100) est une extension de la série globale de normes pour l'Architecture Unifiée OPC et définit les modèles d'information associés aux Appareils. L'IEC 62541-100 décrit trois modèles qui s'appuient les uns sur les autres, comme suit:

- L'objet du Modèle d'appareil (de base) est de fournir une vue unifiée des appareils, quels que soient les protocoles d'appareil sous-jacents.
- Le Modèle de Communication d'Appareil ajoute des éléments d'information de Réseau et de Connexion afin de pouvoir créer les topologies de communication.
- Enfin, le Modèle Device Integration Host (Hôte d'Intégration d'Appareil) ajoute des éléments et des règles supplémentaires exigés pour que les systèmes hôtes gèrent l'intégration pour un système complet. Il permet également de refléter la topologie du système d'automatisation avec les appareils ainsi qu'avec les réseaux de communication connectés.

Le modèle d'information des Appareils spécifie les différents ObjectTypes et d'autres éléments de l'AddressSpace utilisés pour représenter les Appareils et les composants liés, comme l'infrastructure de communication dans un AddressSpace de l'OPC UA. Les principaux cas d'utilisation sont la configuration et le diagnostic d'Appareil, mais ce modèle fournit une méthode générale et normalisée pour permettre aux applications d'accéder aux informations liées à un Appareil.

La Figure 6 représente un exemple de contrôleur de température représenté sous forme d'Objet d'Appareil. Il s'agit d'un Objet DeviceType qui a un sous-type de TopologyElementType et hérite de tous les composants de ce type. Le composant ParameterSet contient toutes les Variables qui décrivent l'Appareil. Le composant MethodSet contient toutes les Méthodes fournies par l'Appareil. Les Composants du type FunctionalGroupType sont utilisés pour regrouper les Paramètres et les Méthodes de l'Appareil dans des groupes logiques. Le FunctionalGroupType et le concept de regroupement sont définis dans l'IEC 62541-100, mais les groupes sont spécifiques au DeviceType, c'est-à-dire que les groupes ProcessData et Configuration sont définis par le TemperatureControllerType dans cet exemple.

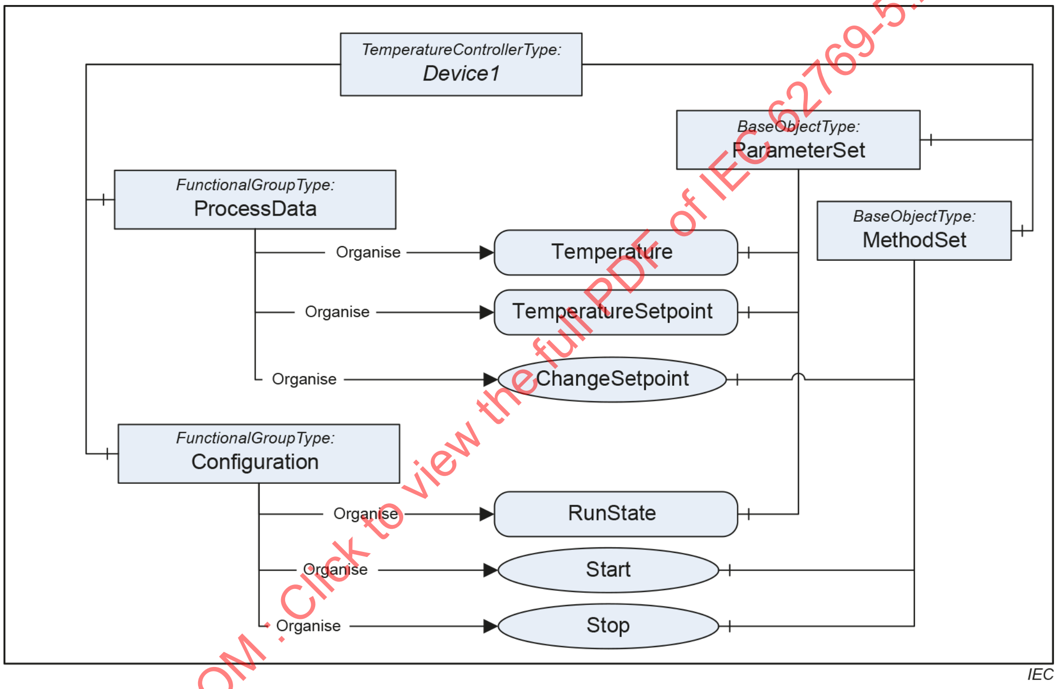


Figure 6 – Exemples d'Appareils OPC UA: Groupes Fonctionnels

Un autre concept défini dans l'IEC 62541-100 est représenté à la Figure 7. Le ConfigurableObjectType est utilisé pour fournir une méthode de regroupement des sous-composants d'un Appareil et pour indiquer quels types de sous-composants peuvent être instanciés. Les types autorisés sont référencés depuis le dossier SupportedTypes. Ces informations peuvent être utilisées par des clients de configuration pour permettre à un utilisateur de choisir le type à instancier en tant que sous-composant de l'Appareil.

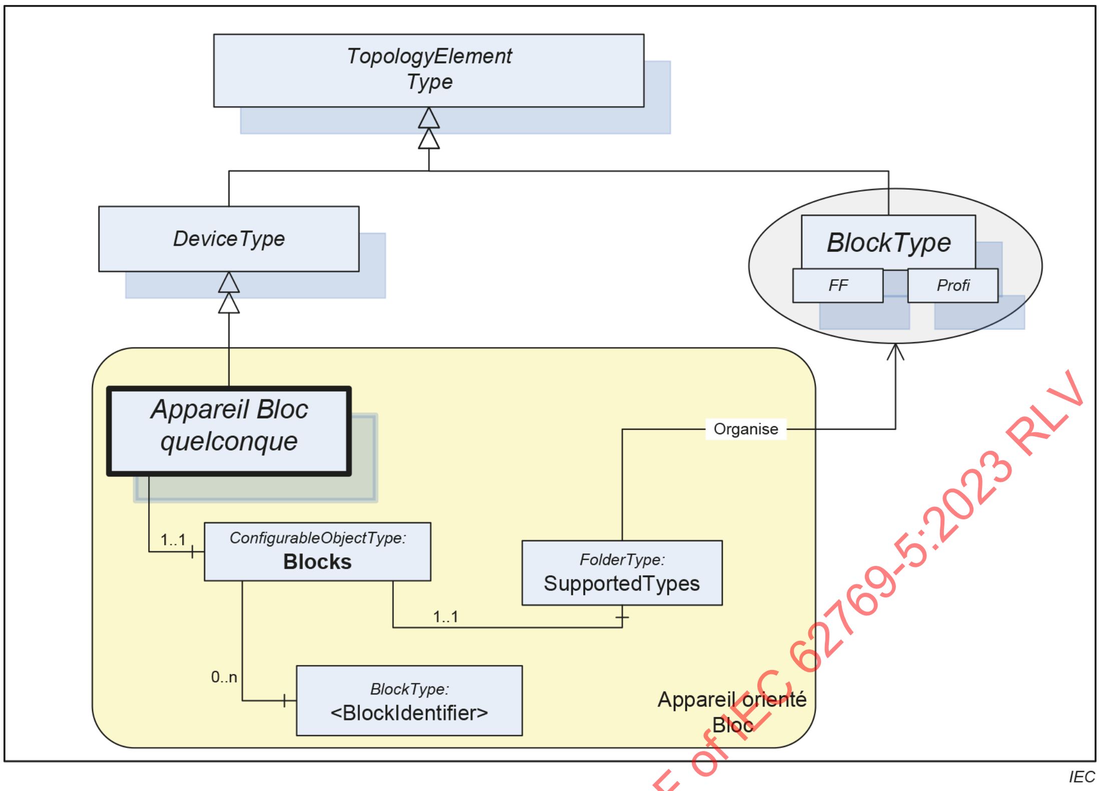


Figure 7 – Exemples d'Appareils OPC UA: Composants configurables

Le dossier SupportedTypes peut contenir différents sous-ensembles d'ObjectTypes pour différentes instances de l'Appareil orienté Bloc en fonction de leur configuration actuelle puisque la liste contient uniquement des types qui peuvent être instanciés pour la configuration actuelle.

5 Concepts

5.1 Généralités

Le Serveur FDI® fournit aux Clients FDI® un accès aux informations relatives aux instances d'Appareils et aux types d'Appareils, indépendamment de l'emplacement de stockage des informations, par exemple dans l'Appareil lui-même ou dans un magasin de données. Ces informations sont fournies par les Services OPC UA et sont appelées Modèle d'Information FDI®.

Le Modèle d'Information FDI® spécifie les entités qui peuvent être accessibles dans le Serveur FDI®, y compris leurs propriétés, leurs relations et les différentes opérations dont elles peuvent faire l'objet. La disponibilité des types d'Appareils ou d'autres éléments topologiques dans un Serveur FDI® donné dépend en grande partie des informations contenues dans les Paquetages FDI®.

5.2 Topologie d'Appareils

L'un des principaux objectifs du Modèle d'Information est de refléter la topologie du système d'automatisation. Le Modèle d'Information représente donc les Appareils du système d'automatisation ainsi que les réseaux de communication de connexion. Le point d'entrée Topologie d'Appareils est le point de départ au sein du Modèle d'Information pour la topologie du système d'automatisation. Le point d'entrée Appareils de Communication contient les appareils de communication utilisés par le Serveur FDI® pour accéder aux éléments de la topologie. La Figure 8 et la Figure 9 représentent un exemple de configuration et de la topologie configurée telle qu'elle apparaît dans l'AddressSpace du Serveur FDI® (omission des informations détaillées).

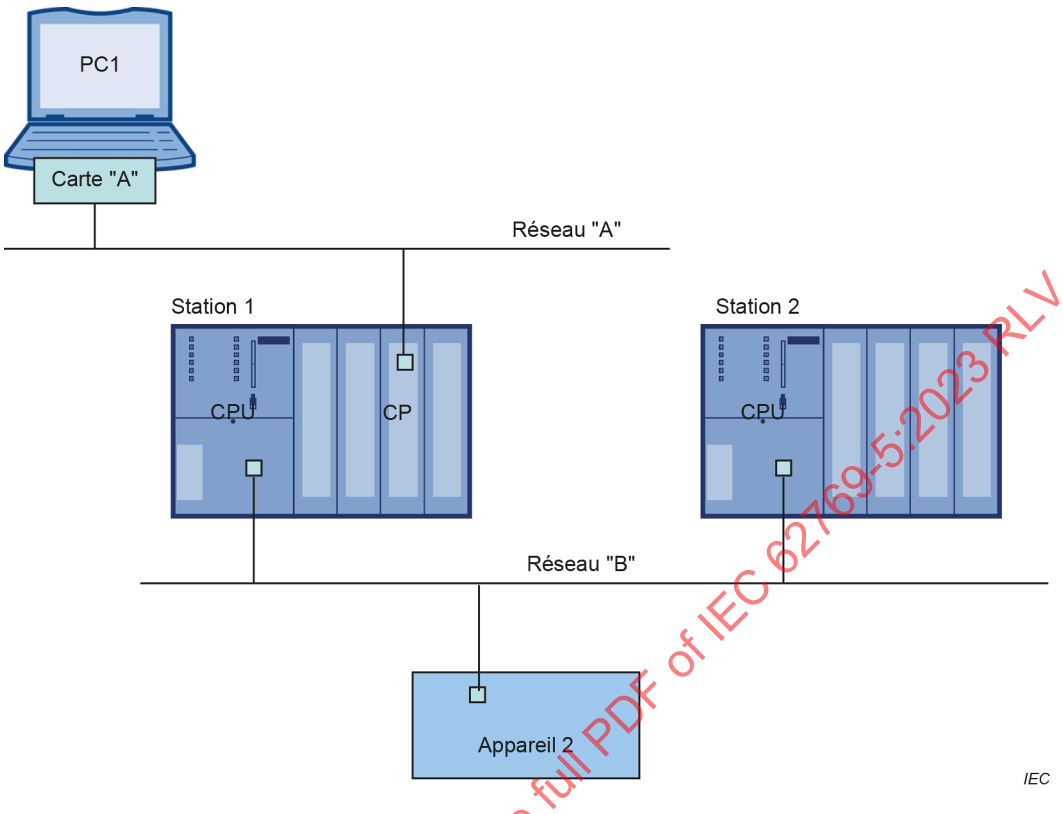


Figure 8 – Exemple de système d'automatisation

Le PC de la Figure 8 représente le boîtier du Serveur FDI®. Le Serveur FDI® communique avec les appareils connectés au Réseau "A" par une Communication Native et avec les appareils connectés au Réseau "B" par une Communication Imbriquée.

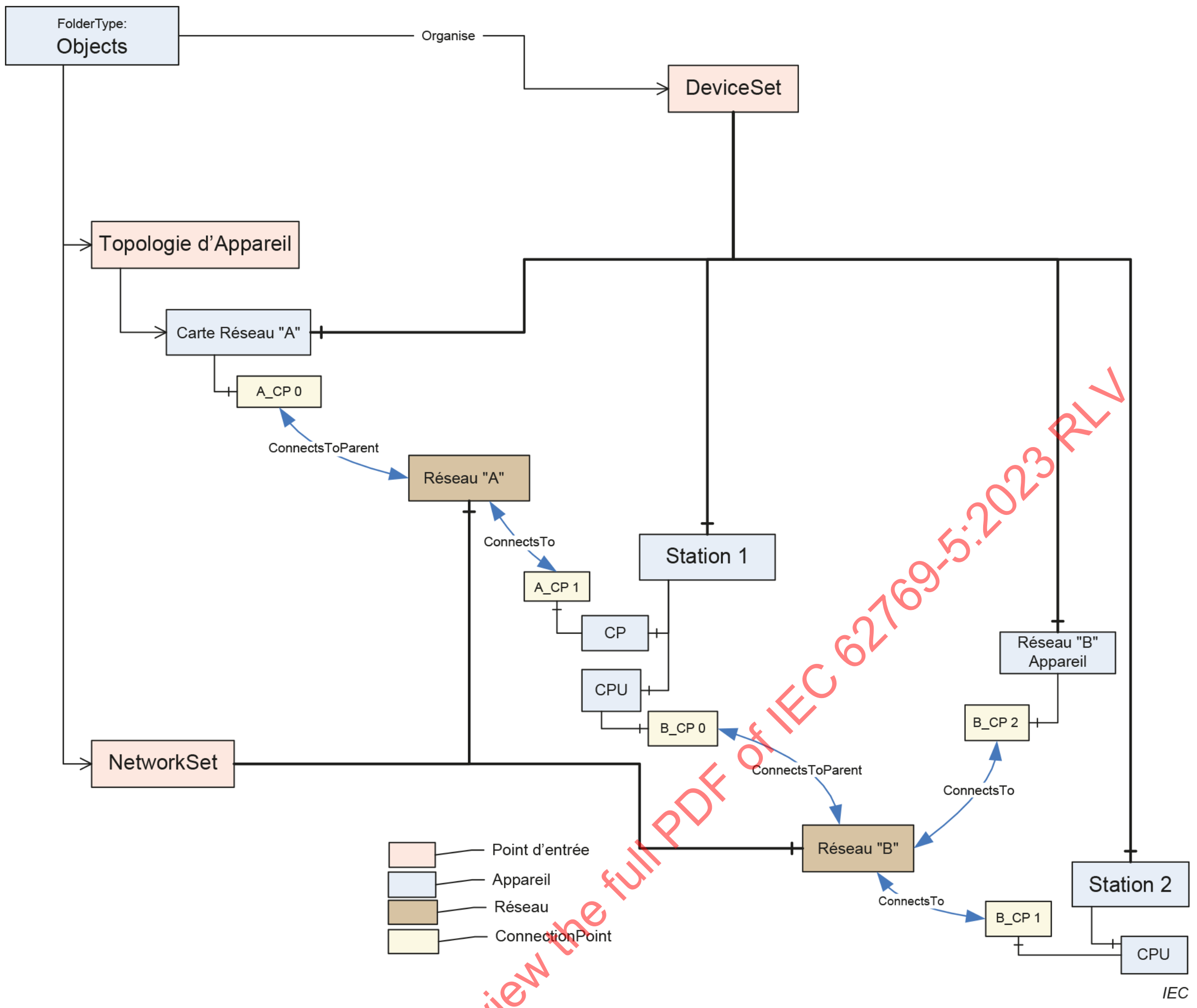


Figure 9 – Exemple de topologie d'appareils

Des rectangles de couleur sont utilisés afin de reconnaître facilement les différents types d'informations.

Les rectangles en marron représentent les réseaux. Les rectangles en bleu clair représentent les Appareils et le jaune clair est utilisé pour les Points de Connexion.

Les rectangles en rose clair représentent les points d'entrée qui assurent un comportement commun à travers différentes mises en œuvre:

- DeviceTopology: nœud de départ pour la configuration de la topologie.
- DeviceSet: tous les Appareils instanciés sont des composants de cet Objet, c'est-à-dire qu'ils existent dans l'AddressSpace indépendamment de la Topologie d'Appareils.
- NetworkSet: tous les Réseaux sont des composants de cet Objet.

5.3 En ligne/Hors-ligne

La gestion de la Topologie d'Appareils est une tâche de configuration, c'est-à-dire que les éléments dans la topologie (Appareils, Réseaux et Points de Connexion) sont généralement configurés "hors-ligne" et sont validés ultérieurement par rapport à leur représentation physique dans un réseau réel.

Afin de prendre en charge l'accès explicite aux informations en ligne ou hors-ligne, chaque élément est représenté par deux instances schématiquement identiques, c'est-à-dire qu'il existe un ParameterSet, des FunctionalGroups, et ainsi de suite. Une Référence relie les représentations en ligne et hors-ligne et permet de naviguer entre elles.

5.4 Catalogue (Définitions de Type)

Les (sous-ensembles de) TopologyElements pris en charge sont organisés en définitions de Type dans l'AddressSpace OPC UA, et forment ainsi un type de catalogue. Ces définitions sont généralement générées à partir d'informations descriptives qui proviennent des Paquetages FDI®. Les définitions de Type contiennent les Paramètres et les valeurs par défaut des Paramètres, Méthodes, Actions et Groupes Fonctionnels, y compris les éléments de l'interface utilisateur. Le Serveur FDI® peut inclure des dossiers dans le modèle de type afin d'organiser les types selon le fabricant ou d'autres critères.

Les définitions de type peuvent alors être utilisées pour créer des instances d'Appareil dans l'AddressSpace OPC UA. Les instances peuvent être créées soit hors-ligne, soit selon des données déterminées par un Balayage. La Figure 10 représente un exemple de certaines définitions de Type (dont les détails sont omis) telles qu'elles peuvent exister dans l'AddressSpace.

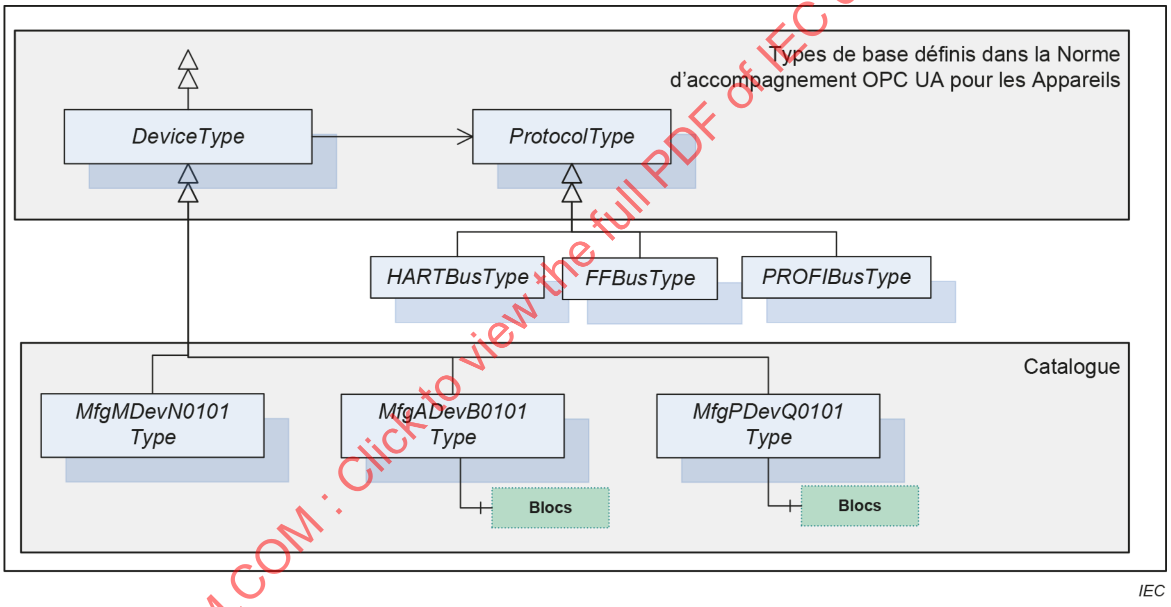


Figure 10 – Exemples de Types d'appareils qui représentent un catalogue

5.5 Communication

Afin d'intégrer des Appareils, le Serveur FDI® nécessite d'être capable de communiquer avec eux. Cette communication est possible au moyen d'une Communication Native ou d'une Communication Imbriquée.

L'exemple donné à la Figure 9 ci-dessus spécifie que le Serveur FDI® a un accès direct au Réseau PROFINET en utilisant sa carte réseau PROFINET. Afin d'accéder à la "Station 2", le Serveur FDI® doit passer par la Station 1 qui fournit les services de communication pour le réseau PROFIBUS DP (voir l'IEC 61784-1-3, CPF 3). Cette procédure peut être effectuée avec une Communication Imbriquée, qui est spécifiée dans l'IEC 62769-3. Les Appareils de Communication et les Serveurs de Communication sont spécifiés dans l'IEC 62769-7.

5.6 Informations sémantiques

Le Système de types de l'OPC UA fournit un moyen d'exprimer la sémantique d'un Objet. Par exemple, la Spécification associée des Appareils définit l'ObjectType DeviceType, qui indique que les instances de cet ObjectType représentent des appareils. Toutefois, la sémantique détaillée de l'appareil n'est pas davantage spécifiée. Les informations sémantiques fournissent un moyen d'indiquer qu'un Objet a la sémantique Transmetteur de pression différentielle, par exemple. Cela permet aux clients de retrouver et d'identifier automatiquement les appareils de ce type.

Dans l'OPC UA, les Objets comportent généralement des Variables qui contiennent des valeurs de variables. Même si le BrowseName ou le DisplayName fournit un "indice" concernant à la sémantique de la Variable, cette méthode ne peut pas être utilisée pour identifier automatiquement la sémantique d'une Variable. Par exemple, le ParameterSet d'un Objet DeviceType contient une liste plate des paramètres de l'appareil. Les informations sémantiques fournissent un moyen d'indiquer qu'une Variable (dans le ParameterSet, par exemple) a une sémantique particulière qui lui est associée. Cela permet aux clients de retrouver et d'identifier automatiquement les Variables de ce type.

L'Objet DictionaryEntryType défini dans l'OPC-10000-19 est utilisé pour ajouter des informations sémantiques aux objets dans le Modèle d'Information FDI®. Les sous-types de DictionaryEntryType doivent définir leur propre espace de noms (http://iec.ch/cdd/iec61987, par exemple). Le NodeId des Objets doit comporter, en tant que partie de l'identificateur, l'indice d'espace de noms correspondant ainsi que la valeur de la Property ID. Cela permet d'accéder directement à l'Objet Semantic sans indirection supplémentaire. L'Annexe A définit les identificateurs numériques de l'ensemble des NodeIds numériques utilisés dans le présent document.

Le Serveur OPC UA doit créer un type d'objet concret dérivé du DictionaryEntryType abstrait pour chaque dictionnaire référencé. Les objets comme DeviceType, FunctionalGroupType ou VariableType peuvent faire référence à un ou plusieurs objets DictionaryEntryType à l'aide de la référence HasDictionaryEntry pour indiquer la valeur sémantique de l'objet. La propriété DictionaryId de l'objet DictionaryEntryType contient la valeur d'identificateur d'une entrée de dictionnaire externe. Un client peut utiliser ces références pour accéder à des données de l'appareil sans qu'il soit nécessaire, d'un point de vue sémantique, de connaître autre chose que les informations souhaitées (autrement dit, aucun objet BrowseName particulier n'est exigé).

L'exemple suivant fournit une vue d'ensemble de ce principe de conception (Figure 11). Le Dictionnaire de données communes (CDD, *Common Data Dictionary*) de l'IEC est utilisé dans cet exemple, mais il convient de noter que tout dictionnaire éligible peut être utilisé. Le Dictionnaire de données communes (CDD) de l'IEC décrit des classes et des propriétés afin de définir les équipements. Par exemple, la série IEC 61987 décrit les équipements d'automatisation de processus. Chaque classe et chaque propriété fournit un ensemble d'attributs (Version, Révision, Nom préférentiel, Définition, etc.). Plus important encore, elle décrit un identificateur propre à chaque classe de propriétés. Dans le cas du CDD de l'IEC, il s'agit d'un Identificateur international de données d'enregistrement (IRDI, *International Registration Data Identifier*). Dans cet exemple, IEC61987DictionalEntryType est défini comme un sous-type de DictionaryEntryType. Les valeurs d'attributs du CDD de l'IEC contenues dans l'Objet DifferentialPressureTransmitter de cet ObjectType sont ses valeurs de propriétés. L'IRDI est défini comme étant la valeur de la propriété ID.

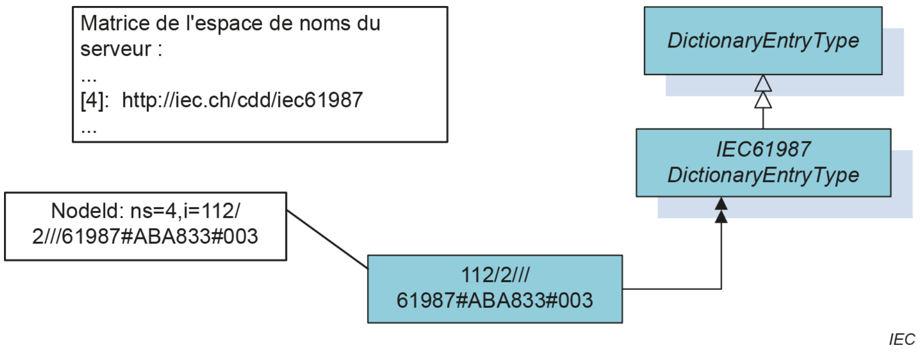

Matrice de l'espace de noms du serveur :
...
[4]: http://iec.ch/cdd/iec61987
...
DictionaryEntryType
IEC61987 DictionaryEntryType
NodeId: ns=4,i=112/ 2///61987#ABA833#003
112/2/// 61987#ABA833#003
IEC

Figure 11 – Exemple de DictionaryEntryType et d'Objet concrets

Les Variables peuvent être des énumérations. Dans ce cas, la Variable peut avoir une sémantique particulière, tout comme chaque élément d'énumération de la Variable. Par exemple, une Variable peut contenir les unités de la valeur de mesure qui peut être configurée. La sémantique de la Variable proprement dite est donc "unité de mesure". Les éléments d'énumération de cette Variable représentent alors une unité de mesure spécifique. Dans le cas d'une Variable de température, le premier élément d'énumération peut représenter des degrés Celsius, le deuxième élément des degrés Fahrenheit, le troisième élément des degrés Kelvin, et ainsi de suite. L'association d'éléments d'énumération à une sémantique particulière permet aux clients de récupérer automatiquement des informations, comme l'unité de mesure actuellement configurée, sans qu'il soit nécessaire de connaître le code d'unité spécifique utilisé par l'appareil.

Les Variables énumérées contiennent une propriété supplémentaire définie en 15.13, DictionaryEntries, qui est une matrice de NodeId de DataType dans laquelle chaque élément contient le NodeId de l'Objet DictionaryEntry correspondant. La Figure 12 donne un exemple: le Parameter2 de la Variable a la Propriété DictionaryEntries. Les éléments de matrice de la Propriété DictionaryEntries contiennent les NodeIds des Objets DictionaryEntry correspondants. Dans cet exemple, Parameter2[0] est lié à l'Objet Semantic1, Parameter2[1] n'est lié à aucun Objet Semantic, et Parameter2[2] est lié à l'Objet Semantic2.

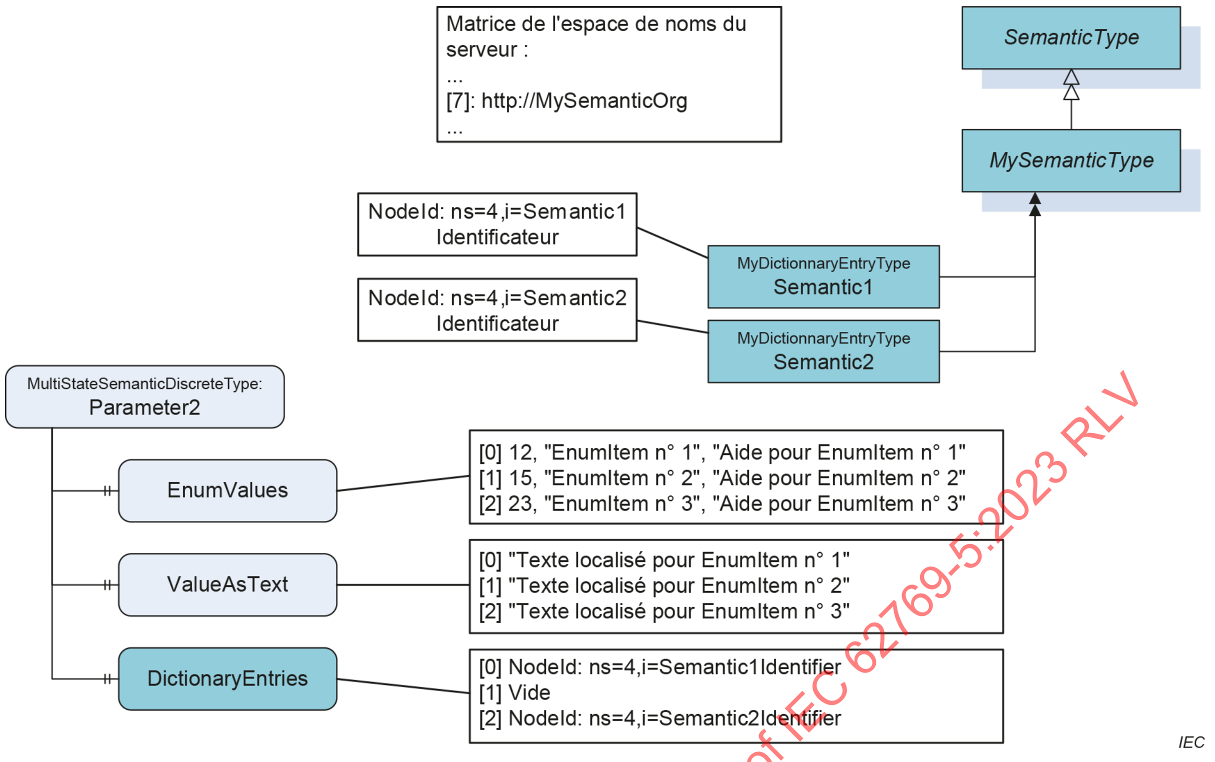

Matrice de l'espace de noms du serveur :
...
[7]: http://MySemanticOrg
...
SemanticType
MySemanticType
NodeId: ns=4,i=Semantic1 Identificateur
NodeId: ns=4,i=Semantic2 Identificateur
MyDictionnaryEntryType Semantic1
MyDictionnaryEntryType Semantic2
MultiStateSemanticDiscreteType: Parameter2
EnumValues
ValueAsText
DictionaryEntries
[0] 12, "EnumItem n° 1", "Aide pour EnumItem n° 1"
[1] 15, "EnumItem n° 2", "Aide pour EnumItem n° 2"
[2] 23, "EnumItem n° 3", "Aide pour EnumItem n° 3"
[0] "Texte localisé pour EnumItem n° 1"
[1] "Texte localisé pour EnumItem n° 2"
[2] "Texte localisé pour EnumItem n° 3"
[0] NodeId: ns=4,i=Semantic1Identifier
[1] Vide
[2] NodeId: ns=4,i=Semantic2Identifier
IEC

Figure 12 – Exemple de DictionaryEntries

La méthode d'interrogation personnalisée GetNodeIdByDictionaryEntryId définie en 15.14 peut être utilisée par un client pour chercher les nœuds qui font référence à un Id d'entrée de dictionnaire spécifié. Cette méthode permet de trouver efficacement des données à l'aide d'informations sémantiques.

6 Organisation de l'AddressSpace

Afin de promouvoir l'interopérabilité des Clients FDI® et des Serveurs FDI®, un ensemble d'Objets et des relations sont définis à l'Article 7 ci-après. Les Serveurs FDI® peuvent mettre en œuvre un sous-ensemble de ces Nœuds normalisés, en fonction de leurs capacités.

Selon les règles de l'OPC UA, un Serveur OPC UA divise l'AddressSpace en deux parties:

- La partie "Types" contient les informations relatives à tous les composants qui ont été générés en fonction des informations descriptives qui proviennent des Paquetages FDI® (voir 5.4).
- La partie "Objets" contient la Topologie d'Appareils avec tous les composants instanciés. Toutes les instances de l'AddressSpace sont reliées à un type du dossier Types.
  – Les points d'entrée DeviceSet, NetworkSet et DeviceTopology sont formellement définis dans l'IEC 62541-100.
    - DeviceTopology est utilisé pour agréger les Réseaux du niveau supérieur qui fournissent un accès à toutes les instances qui constituent la Topologie d'Appareils ((sous-)réseaux, appareils et éléments de communication). Des exemples d'éléments sont donnés ici et sont affichés en vert.
    - Tous les Appareils instanciés sont des composants de l'Objet DeviceSet, c'est-à-dire qu'ils existent dans l'AddressSpace indépendamment de la Topologie d'Appareils. Tous les Réseaux sont des composants de l'Objet NetworkSet.

Les Serveurs FDI® peuvent soit créer des Objets d'Appareil automatiquement, soit montrer uniquement les types disponibles (dossier SupportedTypes) et laisser l'utilisateur créer ses propres instances. Lors de l'utilisation d'une Communication Native, le système fournit généralement la Topologie d'Appareils sans devoir configurer le Client FDI®.

7 Modèle d'Appareil pour FDI®

7.1 Généralités

Comme cela est indiqué précédemment, l'IEC 62541-100 spécifie les types fondamentaux nécessaires pour le FDI®, comme TopologyElementType, DeviceType et ProtocolType (protocole de bus de terrain). L'Article 7 rappelle brièvement les éléments de conception importants spécifiés dans l'IEC 62541-100 et spécifie des types supplémentaires qui ne sont pas mentionnés dans l'IEC 62541-100.

7.2 En ligne/Hors-ligne

Tous les éléments qui apparaissent dans la Topologie d'Appareils (Appareils, Réseaux et Points de Connexion), y compris leur relation, correspondent aux informations stockées dans la base de données de configuration du Serveur FDI®. La gestion de ces éléments exige dans la plupart des cas l'accès au composant/appareil physique (appelé données en ligne dans le présent document), ainsi que le stockage et l'administration des données connexes dans une base de données de configuration (appelées données hors-ligne).

Afin de prendre en charge l'accès explicite aux informations en ligne ou hors-ligne, chaque élément est représenté par deux instances schématiquement identiques, c'est-à-dire qu'il existe un ParameterSet, des FunctionalGroups, et ainsi de suite. Une Référence relie les représentations en ligne et hors-ligne et permet de naviguer entre elles. Cela est représenté à la Figure 13.

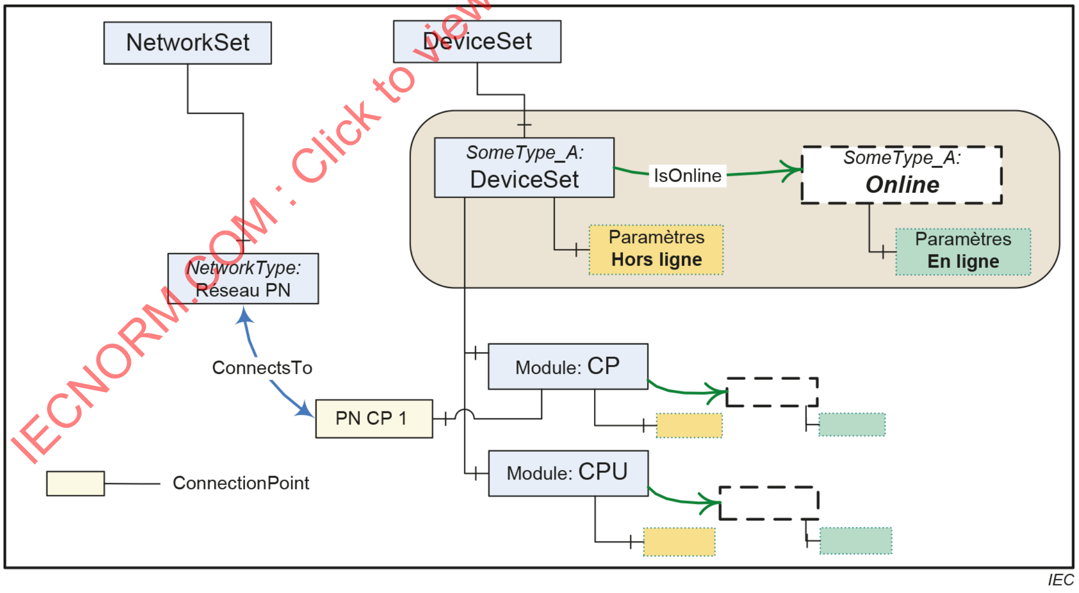


Figure 13 – Composant en ligne pour l'accès aux données d'appareil

La prise en charge en ligne/hors-ligne est obligatoire pour les Serveurs FDI®. Des informations détaillées sur le modèle et les définitions formelles sont spécifiées dans l'IEC 62541-100.

7.3 Santé de l'Appareil

7.3.1 Mapping DeviceHealth

La Propriété DeviceHealth indique le statut d'un appareil comme cela est défini par le document NAMUR NE107. Les Clients FDI® peuvent lire et surveiller cette Propriété afin de déterminer l'état de l'appareil.

Les Serveurs déterminent le statut de santé à l'aide de la METHOD GetHealthStatus de l'EDD, qui est définie dans l'IEC 62769-4. La fréquence à laquelle les Serveurs examinent effectivement le statut de santé peut varier de quelques secondes à quelques minutes.

Le mapping des valeurs de retour de GetHealthStatus à l'OPC UA est spécifié dans le Tableau 1.

Tableau 1 – Mapping DeviceHealth

GetHealthStatus	OPC UA
0 – Indeterminate (Indéterminé)	"Indéterminé" (Indeterminate) n'est pas défini dans le type de données DeviceHealth défini dans l'IEC 62541-100. Les Serveurs qui ne peuvent pas déterminer le statut de santé de l'appareil doivent renvoyer une opération OPC UA Read (Lecture) pour cette Propriété avec un code de statut OPC UA, par exemple: Bad_NotConnected Bad_OutOfService Bad_NoCommunication Bad_NotSupported
Les valeurs suivantes peuvent être mappées aux valeurs correspondantes définies pour le type de données DeviceHealth défini dans l'IEC 62541-100.	
GetHealthStatus	**Valeurs du type de données DeviceHealth**
1 – Failure (Défaillance)	FAILURE_1
2 – Function Check (Vérification fonctionnelle)	CHECK_FUNCTION_2
3 – Out of Specifications (Hors des spécifications)	OFF_SPEC_3
4 – Maintenance Required (Maintenance exigée)	MAINTENANCE_REQUIRED_4
5 – Good (Correct)	GOOD_0

A la différence de l'OPC UA pour les Appareils (IEC 62541-100), la prise en charge de la Propriété DeviceHealth est obligatoire pour les Objets Device dans la FDI®. Cela est représenté au Tableau 2. Le modèle DeviceType complet et sa définition formelle sont spécifiés dans l'IEC 62541-100.

Tableau 2 – Définition de DeviceType (extrait applicable au 7.3.1)

Attribut	**Valeur**				
BrowseName	DeviceType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Le sous-type du TopologyElementType est défini dans l'IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	~~Facultatif~~ → Obligatoire
………					

7.3.2 DeviceHealthDiagnostics

Dans certains cas, un Appareil peut comporter des informations supplémentaires associées au statut de santé, comme la cause possible d'un statut DeviceHealth anormal et des actions suggérées pour rétablir des conditions normales.

Ces informations supplémentaires sont disponibles avec la Variable DeviceHealthDiagnostics. Elle est définie de manière formelle dans le Tableau 3.

Tableau 3 – Définition de DeviceType avec DeviceHealth et DeviceHealthDiagnostics

Attribut	**Valeur**				
BrowseName	DeviceType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Le sous-type du TopologyElementType est défini dans l'IEC 62541-100					
………					
HasComponent	Variable	DeviceHealth	DeviceHealth	BaseDataVariableType	Obligatoire
HasComponent	Variable	DeviceHealthDiagnostics[]	LocalizedText	BaseDataVariableType	Obligatoire
………					

DeviceHealthDiagnostics est une matrice de LocalizedText. Chaque élément de matrice contient des informations de diagnostic associées. La langue doit correspondre au paramètre régional demandé, spécifié dans la Session OPC UA. Voir le 15.2.1 relatif au choix du langage approprié pour la Session OPC UA actuelle.

Lorsque DeviceHealth est demandé, le Serveur doit toujours extraire les DeviceHealthDiagnostics correspondants. Le Serveur doit s'assurer que les valeurs sont synchronisées en exécutant GetHealthStatus dans un premier temps et en lisant ensuite DeviceHealthDiagnostics. Le Serveur met en cache DeviceHealthDiagnostics dans le cadre de la Session OPC UA jusqu'à une nouvelle demande de DeviceHealth.

Si DeviceHealthDiagnostics est lu avec une demande séparée de service, le Serveur doit renvoyer les informations de diagnostic associées avec le statut DeviceHealth lu dernièrement. Si le statut DeviceHealth n'a pas été lu dans la Session OPC UA actuelle, le Serveur doit retourner Bad_NotReadable.

Si aucune information de diagnostic n'est disponible, la Valeur retournée est Null (nulle).

Voir l'IEC 62769-4 concernant la méthode de mapping de DeviceHealthDiagnostics aux informations EDD correspondantes.

Exemple de valeur DeviceHealthInformation avec deux éléments de matrice:

- [0]:
  "Panne d'Alimentation critique\n
  Cause possible: Une panne d'Alimentation critique s'est produite\n
  Action suggérée: Vérifier l'équipement/les environnements/le processus.\n"
- [1]:
  "Dysfonctionnement de l'Appareil\n
  Cause possible: L'appareil a détecté une erreur ou défaillance matérielle majeure.\n
  Action suggérée: Vérifier le diagnostic détaillé de l'appareil si possible et/ou remplacer le matériel de l'appareil si nécessaire.\n"

7.4 Eléments de l'interface utilisateur

7.4.1 Généralités

L'IEC 62541-100 définit de manière abstraite la façon dont les Serveurs peuvent exposer des éléments d'interface utilisateur aux Clients afin d'afficher une interface utilisateur spécifique à un FunctionalGroup d'un TopologyElement.

Le Paragraphe 7.4 spécifie deux types concrets d'éléments utilisateur: les éléments d'interface utilisateur descriptive (les UID) et les éléments d'interface utilisateur programmée (exécutable) (les UIP). Les UIP ne sont jamais référencés directement à partir d'un FunctionalGroup. Ils sont toujours référencés indirectement à partir d'un UID au moyen de leur UipId.

La Figure 14 représente la hiérarchie de types des éléments d'interface utilisateur définis dans l'IEC 62541-100 et dans le présent document.

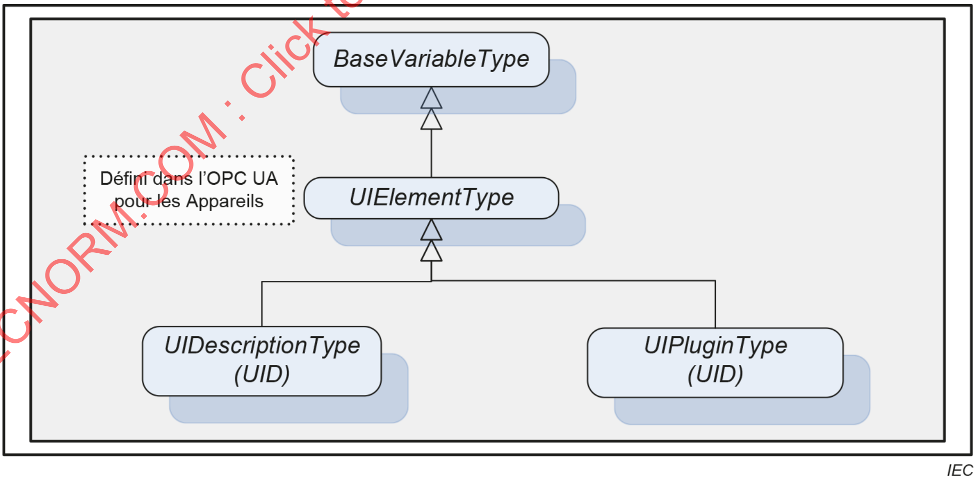


Figure 14 – Hiérarchie des Types d'interfaces utilisateur

7.4.2 Type UI Description

Les Serveurs FDI® peuvent fournir un élément d'interface utilisateur descriptif (un UID) pour chaque FunctionalGroup. Un tel élément est fourni par le Client FDI®. L'UIDescriptionType est formellement spécifié dans le Tableau 4.

Tableau 4 – Définition de l'UIDescriptionType

Attribut	**Valeur**				
BrowseName	UIDescriptionType				
DataType	String				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Hérite des Propriétés de l'UIElementType défini dans l'IEC 62541-100.					

L'Attribut de Valeur fournit l'UID sous forme de Chaîne qui contient un élément XML. Voir l'IEC 62769-2 pour la syntaxe des éléments UID. Le Schéma XML pour la Valeur UID de tous les appareils exposés respecte la même Version de Technologie FDI® que celle qui est indiquée par la Propriété FDIServerVersion (voir Article 14).

7.4.3 Type Plugiciel d'interface utilisateur

Un Plugiciel d'Interface Utilisateur (UIP) est un module logiciel qui est hébergé et exécuté par un Client FDI®. Contrairement à une Description d'interface utilisateur (UID), il s'agit d'un élément d'UI exécutable.

Des détails sur l'hébergement et l'exécution des Plugiciels sont donnés dans l'IEC 62769-2. L'UIPlugInType est formellement spécifié dans le Tableau 5.

Tableau 5 – Définition de l'UIPlugInType

Attribut	**Valeur**				
BrowseName	UIPlugInType				
DataType	Byte				
ValueRank	1 – matrice unidimensionnelle				
ArrayDimensions	Uint32[1] – la longueur (nombre d'octets) de la matrice				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Hérite des Propriétés de l'UIElementType défini dans l'IEC 62541-100.					
HasProperty	Variable	UIPVariantVersion	String	PropertyType	Obligatoire
HasProperty	Variable	FDITechnologyVersion	String	PropertyType	Obligatoire
HasProperty	Variable	RuntimeId	String	PropertyType	Obligatoire
HasProperty	Variable	CpuInformation	String	PropertyType	Obligatoire
HasProperty	Variable	PlatformId	String	PropertyType	Obligatoire
HasProperty	Variable	Style	String	PropertyType	Obligatoire
HasProperty	Variable	StartElementName	String	PropertyType	Obligatoire
HasComponent	Objet	Documentation		FolderType	Facultatif

Les UIP peuvent exister dans de nombreuses variantes pour différentes plateformes ou pour prendre en charge différentes versions. L'UipId (un identificateur unique défini dans le Paquetage FDI®) identifie l'UIP, pas une variante spécifique.

Les UIP n'ont pas à être exposés dans l'AddressSpace. Avec l'UipId, le Client FDI® peut récupérer les NodeId des Variantes d'UIP et leurs Propriétés en appelant le Service TranslateBrowsePathToNodeIds de l'OPC UA avec un NodeId d'Appareil en tant que StartingNode et la liste suivante de noms relatifs:

- "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" est un identificateur du Serveur et ne doit pas forcément être un Nœud dans l'AddressSpace.

Le Serveur FDI® renvoie des matrices de NodeIds pour chaque nom relatif. Le nombre d'entrées dans chaque matrice correspond au nombre de Variantes d'UIP pour l'UipId. Le Client FDI® peut lire les valeurs de propriété en utilisant les NodeId reçus et choisir la Variante UIP appropriée en fonction des FDITechnologyVersion, RuntimeId, CpuInformation et PlatformId.

L'Attribut Value fournit l'exécutable de l'UIP. La représentation exacte dépend de la technologie (voir l'IEC 62769-6). L'Attribut ArrayDimensions doit spécifier la taille (en nombre d'octets) de l'UIP.

Il est nécessaire que les Clients FDI® soient capables de manipuler des UIP de grande taille. Une simple opération Read peut ne pas suffire pour lire des UIP de grande taille, en raison des limites configurées soit dans le Client FDI®, soit dans la pile du Serveur FDI®. La taille maximale par défaut d'une matrice d'octets est de 1 Mo. Les Clients FDI® peuvent utiliser l'IndexRange dans le Service Read de l'OPC UA (voir l'IEC 62541-4) pour lire un UIP en fragments de 1 Mo, par exemple. Le choix de démarrer sans indice et de répéter avec un indexRange uniquement après une erreur ou de toujours utiliser un indexRange relève du Client FDI®.

Les Propriétés suivantes aident le Client FDI® à identifier l'UIP le plus adéquat pour son environnement:

- UIPVariantVersion:
  La version de cette Variante UIP.
- FDITechnologyVersion:
  Version de la Technologie FDI® en fonction de laquelle l'UIP est développé. Un UIP doit toujours être capable de fonctionner dans un système client/serveur avec la même version majeure et une version mineure/de maintenance différente.
- RuntimeId:
  Environnement d'exécution de l'UIP, comme cela est spécifié dans l'IEC 62769-6.
- CpuInformation:
  Fournit les informations supplémentaires à propos de l'environnement d'exécution associé avec le RuntimeId. Les valeurs admises sont spécifiées dans l'IEC 62769-6.

PlatformId définit le type de plateforme qui prend en charge cette Variante d'UIP. Un Client FDI® peut choisir une Variante d'UIP particulière si elle correspond à la plateforme du Client FDI® (voir l'IEC 62769-4 pour les définitions concrètes).

– "Workstation" – avec les résolutions d'écran, les capacités de mémoire et les périphériques d'entrée disponibles (souris et clavier) les plus courants
– "Mobile" – résolution d'écran, mémoire et périphériques d'entrée possibles limités

- Style:
  Détermine si l'UIP doit s'exécuter de manière "modale" ou "non modale", comme cela est défini dans l'IEC 62541-4. Actuellement, les valeurs "Dialog" et "Window" sont définies. "Dialog" exige une fenêtre modale, tandis qu'un UIP de type "Window" est appelé dans une fenêtre modale ou non modale, comme cela est défini dans l'IEC 62769-2.
- StartElementName:
  Elément nécessaire pour démarrer cette Variante d'UIP. L'IEC 62769-6 spécifie la façon dont ces informations sont utilisées lors de l'activation de l'UIP.

Les documents fournis pour une Variante d'UIP sont exposés sous la forme de Variables organisées dans un dossier Documentation. Dans la plupart des cas, ils constituent un manuel de produit, qui peut exister sous la forme d'un ensemble de documents. Ces informations peuvent être récupérées par la lecture de la valeur Variable qui est représentée sous la forme d'une ByteString. La ByteString complète doit être interprétée sous la forme d'un fichier PDF. Il est nécessaire que les Clients FDI® sachent que les contenus représentés par ces variables peuvent être volumineux. Une simple opération Read peut ne pas suffire pour lire des valeurs de grande taille, en raison des limites configurées, soit dans le Client FDI®, soit dans la pile du Serveur FDI®. La taille maximale par défaut d'une matrice d'octets est de 1 Mo. Il est recommandé que les Clients FDI® utilisent l'IndexRange dans le Service Read de l'OPC UA (voir l'IEC 62541-4) pour lire ces Variables en fragments de 1 Mo, par exemple.

7.5 Informations de prise en charge spécifiques au type

Chaque DeviceType peut comporter un ensemble de données complémentaires. Il s'agit principalement d'images, de documents ou de données spécifiques à un protocole. Les différents types d'informations sont organisés en différents dossiers.

Voir l'IEC 62541-100 pour la définition formelle des informations de prise en charge.

7.6 Actions

7.6.1 Vue d'ensemble

Les Actions sont des opérations exécutées dans le Serveur FDI® au nom d'un élément de topologie. Après leur appel avec la Méthode InvokeAction, les Actions peuvent effectuer de nombreuses transitions d'état jusqu'à la fin de la procédure. L'état réel est accessible par l'intermédiaire d'une Variable transitoire et non consultable dont le NodeId est retourné par la Méthode InvokeAction.

Les Clients FDI® peuvent s'abonner à cette Variable pour recevoir les mises à jour concernant l'exécution de l'Action (données d'Action). Les données d'Action peuvent indiquer une transition d'état ou une demande vers le Client FDI® pour lequel l'entrée est nécessaire pour la continuation. Le Client FDI® peut reprendre l'exécution avec la Méthode RespondAction et soumettre les données demandées.

La Figure 15 représente la manière dont les Actions sont intégrées dans un TopologyElement.

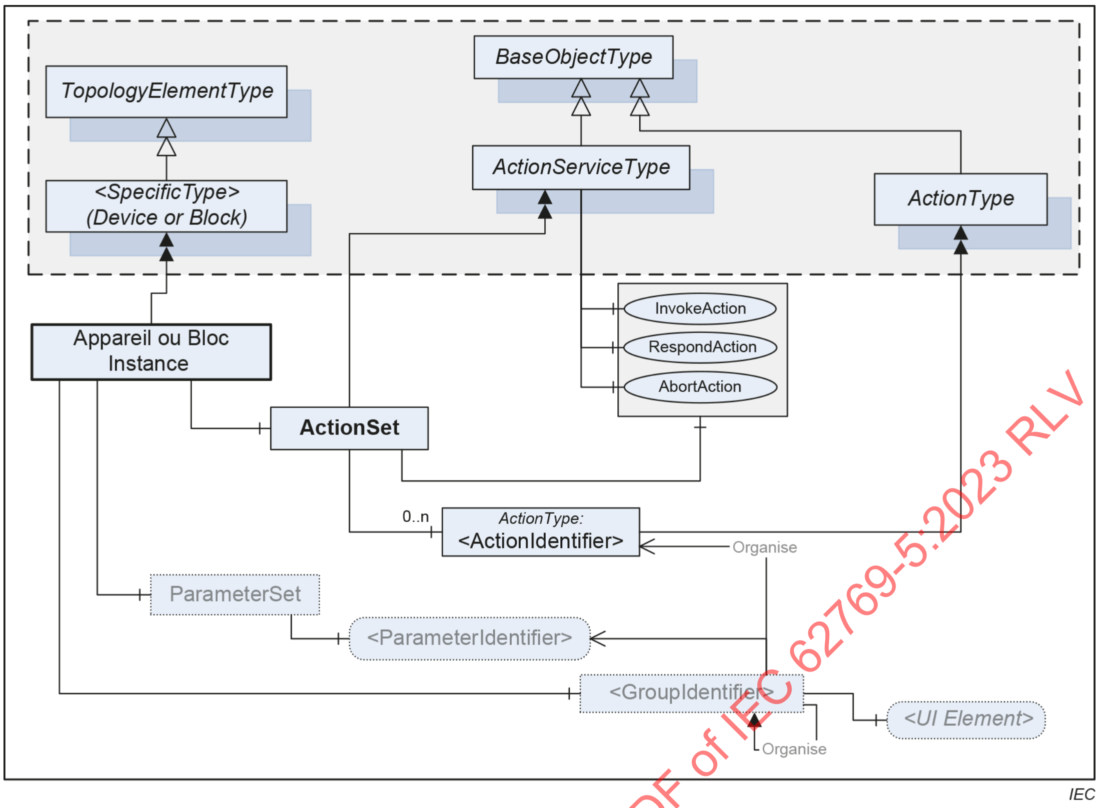


Figure 15 – Intégration d'Actions dans un TopologyElement

ActionSet est utilisé pour agréger les Actions pour un TopologyElement spécifique. Cet Objet n'est pas disponible pour le Type et est uniquement disponible dans une instance lorsqu'il existe des Actions pour ce TopologyElement.

Les Actions peuvent également être référencées à partir des Objets FunctionalGroup.

7.6.2 Type Action

Cet ObjectType définit la structure d'une Action. Il est défini de manière formelle dans le Tableau 6.

Tableau 6 – Définition de l'ActionType

Attribut	**Valeur**				
BrowseName	ActionType				
IsAbstract	True				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Le sous-type du BaseObjectType est défini dans l'IEC 62541-5.					

Le Client FDI® détermine les arguments d'appel exigés pour une Action depuis l'UID.

7.6.3 Type ActionService

L'ActionServiceType définit les Méthodes pour appeler et commander les Actions. Les instances de ce type agrègent les Actions pour un élément de topologie spécifique. L'ActionServiceType est défini de manière formelle dans le Tableau 7. Son utilisation est représentée à la Figure 15.

Tableau 7 – Définition de l'ActionServiceType

Attribut	Valeur				
BrowseName	ActionServiceType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du BaseObjectType défini dans l'IEC 62541-5.					
HasComponent	Méthode	InvokeAction			Obligatoire
HasComponent	Méthode	RespondAction			Obligatoire
HasComponent	Méthode	AbortAction			Obligatoire
HasComponent	Objet	<ActionIdentifier>		ActionType	Facultatif

L'ActionServiceType et chaque instance de ce Type partagent les mêmes Méthodes. Le NodeId de ces Méthodes est établi et défini dans le présent document. Les Clients FDI® n'ont donc pas à rechercher ces Méthodes. Ils peuvent utiliser les NodeId fixés comme MethodId du Service d'appel.

Le StatusCode de l'OPC UA, Bad_MethodInvalid, doit être retourné depuis le Service Call pour les éléments pour lesquels les Méthodes ActionService ne sont pas prises en charge.

<ActionIdentifier> désigne une ou plusieurs Actions. Les Actions (Objets de l'ActionType) existent uniquement dans des instances de l'ActionServiceType, c'est-à-dire lorsqu'une instance de cet ObjectType est ajoutée à un TopologyElement. Il n'existe pas d'Actions (Objets de l'ActionType) dans l'ActionServiceType lui-même.

7.6.4 Objet ActionService

La prise en charge d'ActionService pour un Objet est déclarée par l'agrégation d'une instance du Type ActionService, comme cela est représenté à la Figure 16.

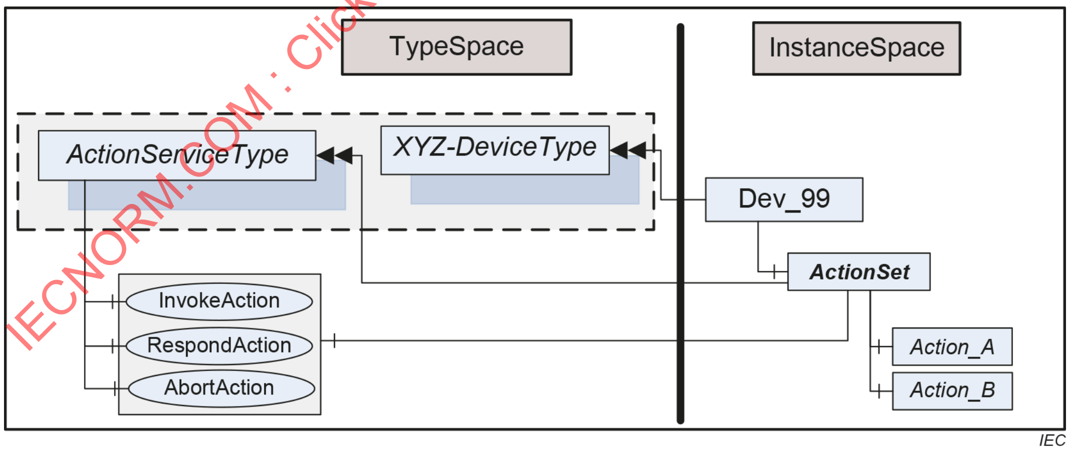


Figure 16 – Service Action

Cet Objet est utilisé en tant que conteneur pour les Méthodes ActionService et doit avoir le BrowseName ActionSet. Il est défini de manière formelle dans le Tableau 7. HasComponent est utilisé comme référence d'un TopologyElement (par exemple, un Appareil) à son Objet "ActionService".

L'ActionServiceType et chaque Objet ActionSet partagent les mêmes Méthodes. Les Actions sont généralement partagées par toutes les instances du même Type d'Appareil.

7.6.5 Méthode InvokeAction

InvokeAction est utilisé pour lancer une Action. Il retourne immédiatement un résultat après la création du diagramme d'états. Le composant "ActionSet" du TopologyElement (Appareil) au nom duquel l'Action doit être appelée est spécifié par l'argument ObjectId du Service Call.

Un verrouillage explicite est exigé. Si l'Appareil n'a pas été verrouillé, le Serveur FDI® rejette la demande.

Lorsque que la Méthode InvokeAction retourne un résultat, le Client FDI® doit s'abonner à l'Attribut Value de l'ActionNodeId. L'Attribut Value contient un élément XML (DataType = String (Chaîne)) qui reflète l'état actuel de l'Action, ainsi que des données supplémentaires (en fonction de l'état). Le Client FDI® obtient alors une notification DataChange à chaque variation d'état de l'Action.

Voir l'IEC 62769-2 pour les diagrammes d'états de l'Action ainsi que pour le Schéma XML de la valeur ActionNodeId.

Les Serveurs FDI® doivent mettre en mémoire cache les données d'états de l'Action pendant une durée appropriée (quelques secondes) afin qu'aucune information d'états ne soit perdue jusqu'à ce que le Client FDI® ait l'occasion de s'abonner.

La signature de cette Méthode est spécifiée ci-dessous. Le Tableau 8 et le Tableau 9 spécifient les arguments et la représentation AddressSpace, respectivement.

Signature

```
InvokeAction(
  [in]  String      ActionName,
  [in]  String      MethodArguments,
  [out] NodeId      ActionNodeId,
  [out] Int32       InvokeActionError);
```

Tableau 8 – Arguments de la Méthode InvokeAction

Argument	Description
ActionName	Portion String (chaîne) du BrowseName de l'Action qui est utilisée dans l'instance ActionServiceType.
MethodArguments	Document XML qui contient les arguments d'entrée de l'Action (le cas échéant). Le Schéma XML ListOfActionArguments défini dans l'IEC 62769-2 est utilisé pour le paramètre MethodArguments.
ActionNodeId	Nœud non consultable du type Variable. Ce nœud est utilisé à la fois pour identifier l'instance du diagramme d'état de l'Action et pour accéder aux informations d'états de l'Action.
InvokeActionError	0 – OK. -1 – E_LockRequired – l'élément n'est pas verrouillé comme cela est exigé -2 – E_UnknownAction – le nom transmis n'est pas une action valide pour cet élément

Tableau 9 – Définition de l'AddressSpace de la Méthode InvokeAction

Attribut	Valeur				
BrowseName	InvokeAction				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.6 Méthode RespondAction

La Méthode RespondAction est utilisée par le Client FDI® pour fournir la valeur demandée ou la sélection d'une demande de fonction de l'UI (le cas échéant). Le composant "ActionSet" du TopologyElement au nom duquel l'Action a été appelée est spécifié par l'argument ObjectId du Service Call.

La signature de cette Méthode est spécifiée ci-dessous. Le Tableau 10 et le Tableau 11 spécifient les arguments et la représentation AddressSpace, respectivement.

Signature

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

Tableau 10 – Arguments de la Méthode RespondAction

Argument	Description
ActionNodeId	NodeId d'une Variable transitoire qui représente et identifie l'Action en cours d'exécution. Cet Id est retourné par la Méthode InvokeAction.
Response	Document XML qui contient la réponse (valeur ou sélection). Voir l'IEC 62769-2 pour le Schéma XML du paramètre Response.
RespondActionError	0 – OK -1 – E_InvalidAction – le NodeId ne fait pas référence à une action existante -2 – E_InvalidResponse – les données de réponse transmises n'ont pas pu être interprétées

Tableau 11 – Définition de l'AddressSpace de la Méthode RespondAction

Attribut	Valeur				
BrowseName	RespondAction				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.7 Méthode AbortAction

AbortAction est utilisé par le Client FDI® pour abandonner l'exécution d'une Action. L'appel de cette Méthode retourne immédiatement un résultat. Le Client FDI® est informé par un événement de notification sur la Variable ActionNodeId lorsque l'Action se met dans l'état Aborting (Abandon).

Le composant "ActionSet" du TopologyElement au nom duquel l'Action a été appelée est spécifié par l'argument ObjectId du Service Call.

La signature de cette Méthode est spécifiée ci-dessous. Le Tableau 12 et le Tableau 13 spécifient les arguments et la représentation AddressSpace, respectivement.

Signature

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

Tableau 12 – Arguments de la Méthode AbortAction

Argument	Description
ActionNodeId	NodeId d'une Variable transitoire qui représente et identifie l'Action en cours d'exécution. Cet Id est retourné par la Méthode InvokeAction.
AbortActionError	0 – OK -1 – E_InvalidAction – le NodeId ne fait pas référence à une action existante

Tableau 13 – Définition de l'AddressSpace de la Méthode AbortAction

Attribut	Valeur				
BrowseName	AbortAction				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	Variable	InputArguments	Argument[]	PropertyType	Obligatoire
HasProperty	Variable	OutputArguments	Argument[]	PropertyType	Obligatoire

7.6.8 Transfert interactif vers l'appareil

La fonctionnalité qui permet de transférer des données vers l'Appareil en ligne, ce qui inclut les interactions utilisateur, est fournie sous forme d'ACTION INTERACTIVE_TRANSFER_TO_DEVICE par le Serveur FDI®. Cette action ne comporte aucun argument. Elle est démarrée à l'aide de la méthode InvokeAction.

8 Réseau et connectivité

Les éléments d'information de réseau et de connexion sont exigés pour créer les topologies de communication.

Un Réseau représente les moyens de communication pour les Appareils auxquels il est connecté. Il comprend les technologies filaires et sans fil. Les ConnectionPoints représentent l'interface (carte d'interface) d'un Appareil à un Réseau. Un sous-type spécifique doit être défini pour chaque protocole.

Ces éléments sont décrits et formellement définis dans l'IEC 62541-100.

9 Fonctions utilitaires

9.1 Vue d'ensemble

L'Article 9 fournit des services spécifiques pour certains éléments d'information de FDI®.

9.2 Locking

Le Locking (verrouillage) est le moyen d'éviter des modifications simultanées sur un Appareil ou un Réseau et leurs composants. Les Clients FDI® doivent utiliser les Services de Verrouillage pour toute modification (opérations d'écriture ou appels d'Actions, par exemple).

L'objectif principal du verrouillage d'un Appareil est d'éviter les modifications d'appareil simultanées. L'objectif principal du verrouillage d'un Réseau est d'éviter les modifications de topologie simultanées.

Lors du verrouillage d'un Appareil, le verrou s'applique toujours aux versions en ligne et hors-ligne.

Lors du verrouillage d'un Appareil Modulaire, le verrou s'applique à l'Appareil complet (y compris tous les modules). De la même manière, lors du verrouillage d'un Appareil Bloc, le verrou s'applique à l'Appareil complet (y compris tous les blocs).

Si aucun verrou n'est appliqué à l'Appareil de niveau supérieur (pour Appareil Modulaire ou Appareil Bloc), les sous-appareils ou les blocs, respectivement, peuvent être verrouillés indépendamment.

Lors du verrouillage d'un Réseau, le verrou s'applique au Réseau et à tous les Appareils connectés. Si un des Appareils connectés fournit un accès à un Réseau subordonné (comme une Passerelle), le Réseau subordonné et ses Appareils connectés sont également verrouillés.

Le LockingService est entièrement décrit et formellement défini dans l'IEC 62541-100.

9.3 EditContext

9.3.1 Vue d'ensemble

Un Editcontext peut être utilisé pour apporter des modifications aux valeurs Variable visibles du Serveur sans les appliquer à l'Appareil. Le Serveur FDI® fournit le concept EditContext pour prendre en charge les Clients dans leur tâche d'édition.

L'EditContext est spécifié dans l'IEC 62769-3. Le Modèle d'Information OPC UA, y compris les Méthodes pour gérer les instances EditContext, est défini ci-dessous.

EditContext est exposé sous la forme d'une capacité AddIn, qui est comparable à la technologie d'interface présente dans certains langages de programmation. Le service Editcontext est modélisé sous la forme d'un ObjectType et des instances de ce type sont ajoutées à l'Appareil avec un BrowseName prédéfini. Des exemples d'AddIn supplémentaires sont définis dans l'IEC 62541-100.

Lors de la lecture ou de l'abonnement aux valeurs Variable enregistrées dans un EditContext, le StatusCode suivant spécifique au FDI® peut se produire (voir Article 11):

- Good_Edited distingue les valeurs qui ont été éditées, mais qui n'ont pas été écrites sur l'Appareil.
- Uncertain_DominantValueChanged indique que des valeurs dépendantes ne sont pas valides et sont recalculées après application de la valeur dominante à l'appareil. Dans le cas d'un appareil hors-ligne, la valeur dépendante doit également être écrite par le Client.
- Good_DependentValueChanged indique qu'une valeur dépendante a été modifiée, mais sans appliquer la modification à l'appareil.