IEC 62541-8

Edition 3.0 2020-06
REDLINE VERSION

INTERNATIONAL STANDARD

colour
inside

OPC unified architecture –
Part 8: Data access

IEC 62541-8:2020-06 RLV(en)

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

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

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

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

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 It also gives information on projects, replaced and withdrawn publications.

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

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

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC 62541-8

Edition 3.0 2020-06
REDLINE VERSION

INTERNATIONAL STANDARD

colour
inside

OPC unified architecture –
Part 8: Data access

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.05

ISBN 978-2-8322-8572-5

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

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

OPC UNIFIED ARCHITECTURE –

Part 8: Data access

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.

3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

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

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.

8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

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

International Standard IEC 62541-8 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2015. This edition constitutes a technical revision.

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

a) added new VariableTypes for AnalogItems;

b) added an Annex that specifies a recommended mapping of OPC UA Dataccess to OPC COM DataAccess;

c) changed the ambiguous description of "Bad_NotConnected";

d) updated description for EUInformation to refer to latest revision of UNCEFACT units.

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

FDIS	Report on voting
65E/708/FDIS	65E/726/RVD

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

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

Throughout this document and the other parts of the IEC 62541 series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in the "Terms and definition" clause in one of the parts of the IEC 62541 series.

Italics are also used to denote the name of a service input or output parameter or the name of a structure or element of a structure that are usually defined in tables.

The *italicized terms and names* are, with a few exceptions, written in camel-case (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). For example, the defined term is *AddressSpace* instead of Address Space. This makes it easier to understand that there is a single definition for *AddressSpace*, not separate definitions for Address and Space.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, 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 "http://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 publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

OPC UNIFIED ARCHITECTURE –

Part 8: Data access

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture (OPC UA) standard series and defines the information model associated with Data Access (DA). It particularly includes additional *VariableTypes* and complementary descriptions of the *NodeClasses* and *Attributes* needed for Data Access, additional *Properties,* and other information and behaviour.

The complete address space model, including all *NodeClasses* and *Attributes* is specified in IEC 62541-3. The services to detect and access data are specified in IEC 62541-4.

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 TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

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*

UN/CEFACT: UNECE Recommendation N° 20, *Codes for Units of Measure Used in International Trade*, available at https://www.unece.org/cefact/codesfortrade/codes_index.html

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC 62541-3, and IEC 62541-4 and the following apply.

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

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

3.1.1
DataItem
link to arbitrary, live automation data, that is, data that represents currently valid information

Note 1 to entry: Examples of such data are

- device data (such as temperature sensors),

- calculated data,
- status information (open/closed, moving),
- dynamically changing system data (such as stock quotes),
- diagnostic data.

3.1.2
AnalogItem
DataItem that represents continuously variable physical quantities (e.g. length, temperature), in contrast to the digital representation of data in discrete items

Note 1 to entry: Typical examples are the values provided by temperature sensors or pressure sensors. OPC UA defines a specific *VariableType* to identify an *AnalogItem*. *Properties* describe the possible ranges of *AnalogItem*s.

3.1.3
DiscreteItem
DataItem that represents data that may take on only a certain number of possible values (e.g. OPENING, OPEN, CLOSING, CLOSED)

Note 1 to entry: Specific *VariableTypes* are used to identify *DiscreteItems* with two states or with multiple states. *Properties* specify the string values for these states.

3.1.4
ArrayItem
DataItem that represents continuously variable physical quantities and where each individual data point consists of multiple values represented by an array (e.g., the spectral response of a digital filter)

Note 1 to entry: Typical examples are the data provided by analyser devices. Specific *VariableTypes* are used to identify *ArrayItem* variants.

3.1.5
EngineeringUnits
units of measurement for *AnalogItems* that represent continuously variable physical quantities (e.g. length, mass, time, temperature)

Note 1 to entry: This standard defines *Properties* to inform about the unit used for the *DataItem* value and about the highest and lowest value likely to be obtained in normal operation.

3.2 Abbreviated terms ~~and symbols~~

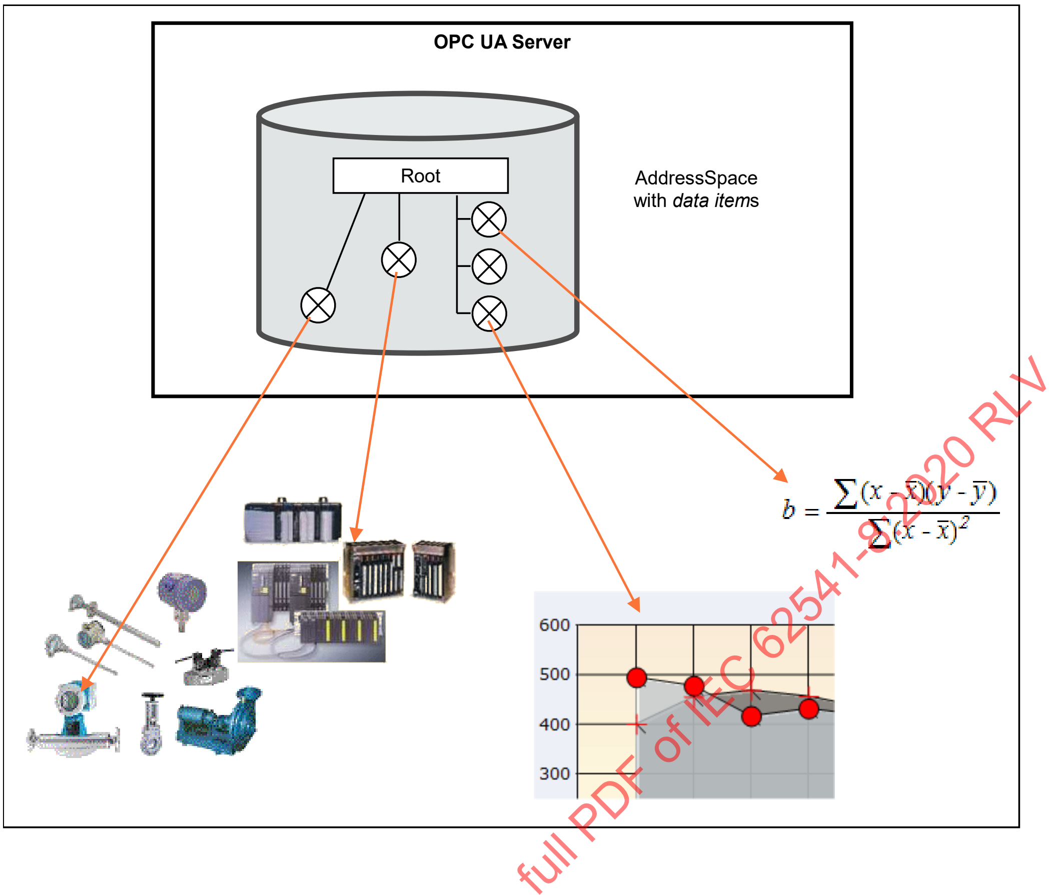
DA	data access
EU	engineering unit
UA	Unified Architecture

4 Concepts

Data Access deals with the representation and use of automation data in Servers.

Automation data can be located inside the *Server* or on I/O cards directly connected to the *Server*. It can also be located in sub-servers or on other devices such as controllers and input/output modules, connected by serial links via field buses or other communication links. OPC UA Data Access *Servers* provide one or more OPC UA Data Access *Clients* with transparent access to their automation data.

The links to automation data instances are called *DataItems*. The categories of automation data are provided is completely vendor-specific. Figure 1 illustrates how the *AddressSpace* of a *Server* ~~might~~ may consist of a broad range of different *DataItem*s.


OPC UA Server
Root
AddressSpace
with *data items*
$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$
600
500
400
300

IEC

Figure 1 – OPC *DataItems* are linked to automation data

Clients may read or write *DataItems*, or monitor them for value changes. The *Services* needed for these operations are specified in IEC 62541-4. Changes are defined as a change in status (quality) or a change in value that exceeds a client-defined range called a *Deadband*. To detect the value change, the difference between the current value and the last reported value is compared to the *Deadband*.

5 Model

5.1 General

The DataAccess model extends the variable model by defining *VariableTypes*. The *DataItemType* is the base type. *ArrayItemType*, ~~*AnalogItemType*~~ *BaseAnalogType* and *DiscreteItemType* ~~(and its *TwoState* and *MultiState* subtypes)~~ are specializations. See Figure 2. Each of these *VariableTypes* can be further extended to form domain- or server-specific *DataItems*.

Annex A specifies the recommended way for mapping the information received from OPC COM Data Access (DA) Servers to the model in this document.

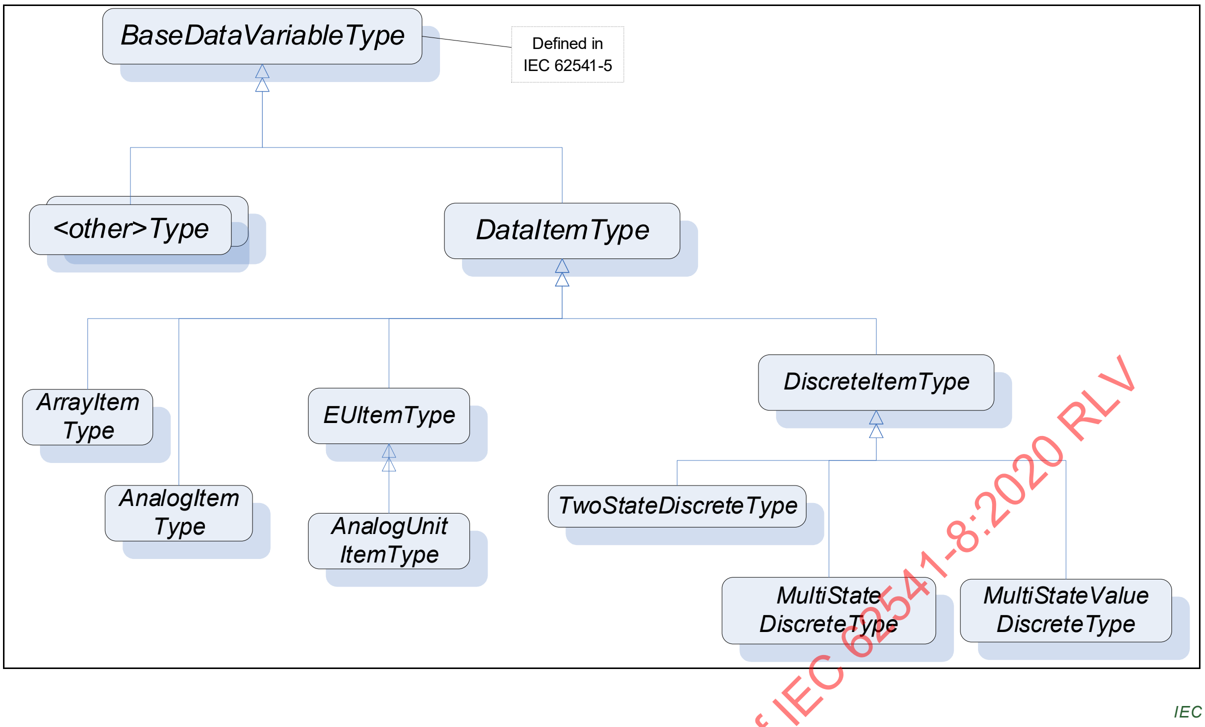


Figure 2 – DataItem *VariableType* hierarchy

5.2 SemanticsChanged

The *StatusCode* also contains an informational bit called *SemanticsChanged*.

Servers that implement Data Access shall set this Bit in notifications if certain *Properties* defined in this standard change. The corresponding *Properties* are specified individually for each *VariableType*.

Clients that use any of these *Properties* should re-read them before they process the data value.

5.3 Variable Types

5.3.1 DataItemType

This *VariableType* defines the general characteristics of a *DataItem*. All other *DataItem* Types derive from it. The *DataItemType* derives from the *BaseDataVariableType* and therefore shares the variable model as described in IEC 62541-3 and IEC 62541-5. It is formally defined in Table 1.

Table 1 – DataItemType definition

Attribute	**Value**				
BrowseName	DataItemType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseDataVariableType* defined in IEC 62541-5; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogItemType	Defined in 5.3.2		
HasSubtype	VariableType	DiscreteItemType	Defined in 5.3.3		
HasSubtype	VariableType	ArrayItemType	Defined in 5.3.4		
HasProperty	Variable	Definition	String	PropertyType	Optional
HasProperty	Variable	ValuePrecision	Double	PropertyType	Optional

Definition is a vendor-specific, human-readable string that specifies how the value of this *DataItem* is calculated. *Definition* is non-localized and will often contain an equation that can be parsed by certain clients.

EXAMPLE: *Definition*::= "(TempA – 25) + TempB"

ValuePrecision specifies the maximum precision that the *Server* can maintain for the item based on restrictions in the target environment.

ValuePrecision can be used for the following *DataTypes*:

- for Float and Double values it specifies the number of digits after the decimal place;
- for DateTime values it indicates the minimum time difference in nanoseconds. For example, a ValuePrecision of 20 000 000 defines a precision of 20 ms.

The *ValuePrecision Property* is an approximation that is intended to provide guidance to a *Client*. A *Server* is expected to silently round any value with more precision that it supports. This implies that a *Client* may encounter cases where the value read back from a *Server* differs from the value that it wrote to the *Server*. This difference shall be no more than the difference suggested by this *Property*.

5.3.2 AnalogItem VariableTypes

~~This *VariableType* defines the general characteristics of an *AnalogItem*. All other *AnalogItem* Types derive from it. The *AnalogItemType* derives from the *DataItemType*. It is formally defined in Table 2.~~

5.3.2.1 General

The *VariableTypes* in this subclause define the characteristics of *AnalogItems*. The types have identical semantics and *Properties* but with diverging *ModellingRules* for individual *Properties*.

The *Properties* are only described once – in 5.3.2.2. The descriptions apply to the *Properties* for the other *VariableTypes* as well.

5.3.2.2 BaseAnalogType

This *VariableType* is the base type for analog items. All *Properties* are optional. Subtypes of this base type will mandate some of the *Properties*. The *BaseAnalogType* derives from the *DataItemType*. It is formally defined in Table 2.

Table 2 – ~~*AnalogItemType*~~ BaseAnalogType definition

Attribute	Value				
BrowseName	~~AnalogItemType~~ BaseAnalogType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.3.1 i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogItemType	Defined in 5.3.2.3		
HasSubtype	VariableType	AnalogUnitType	Defined in 5.3.2.4		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	~~Mandatory~~Optional
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional

The following paragraphs describe the *Properties* of this *VariableType*. If the analog item's *Value* contains an array, the *Properties* shall apply to all elements in the array.

InstrumentRange defines the value range that can be returned by the instrument.

EXAMPLE 1 *InstrumentRange*::= {−9 999,9, 9 999,9}

Although defined as optional, it is strongly recommended for *Servers* to support this *Property*. Without an *InstrumentRange* being provided, *Clients* will commonly assume the full range according to the *DataType*.

The *InstrumentRange Property* may also be used to restrict a Built-in DataType such as Byte or Int16) to a smaller range of values.

EXAMPLE 2

Uint4:	*InstrumentRange*::= {0, 15}
Int6:	*InstrumentRange*::= {−32, 31}

The *Range Data Type* is specified in 5.6.2.

EURange defines the value range likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.

Sensor or instrument failure or deactivation can result in a returned item value which is actually outside of this range. *Client* software ~~must~~ shall be prepared to deal with this possibility. Similarly a *Client* may attempt to write a value that is outside of this range back to the server. The exact behaviour (accept, reject, clamp, etc.) in this case is *Server*-dependent. However, in general *Servers* shall be prepared to handle this.

EXAMPLE 3 *EURange*::= {−200,0, 1 400,0}

See also 6.2 for a special monitoring filter (*PercentDeadband*) which is based on the engineering unit range.

NOTE 1 If *EURange* is not provided on an instance, the *PercentDeadband* filter cannot be used for that instance (see 6.2).

EngineeringUnits specifies the units for the *DataItem*'s value (e.g., DEGC, hertz, seconds). The *EUInformation* type is specified in 5.6.3.

It is important to note that understanding the units of a measurement value is essential for a uniform system. In an open system in particular where *Servers* from different cultures might be used, it is essential to know what the units of measurement are. Based on such knowledge, values can be converted if necessary before being used. Therefore, although defined as optional, support of the *EngineeringUnits Property* is strongly advised.

OPC UA recommends using the "Codes for Units of Measurement" (see UN/CEFACT: UNECE Recommendation N° 20). The mapping to the *EngineeringUnits Property* is specified in 5.6.3.

NOTE 2 Examples for unit mixup: in 1999, the Mars Climate Orbiter crashed into the surface of Mars. The main reason was a discrepancy over the units used. The navigation software expected data in newton second; the company who built the orbiter provided data in pound-force seconds. Another, less expensive, disappointment occurs when people used to British pints order a pint in the USA, only to be served what they consider a short measure.

The *StatusCode SemanticsChanged* bit shall be set if any of the *EURange* (could change the behaviour of a *Subscription* if a *PercentDeadband* filter is used) or *EngineeringUnits* (could create problems if the *Client* uses the value to perform calculations) *Properties* are changed (see 5.2 for additional information).

5.3.2.3 AnalogItemType

This *VariableType* requires the *EURange Property*. The *AnalogItemType* derives from the *BaseAnalogType*. It is formally defined in Table 3.

Table 3 – AnalogItemType definition

Attribute	**Value**				
BrowseName	AnalogItemType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseAnalogType* defined in 5.3.2.2, i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogUnitRangeType	Defined in 5.3.2.5		
HasProperty	Variable	EURange	Range	PropertyType	Mandatory

5.3.2.4 AnalogUnitType

This *VariableType* requires the *EngineeringUnits Property*. The *AnalogUnitType* derives from the *BaseAnalogType*. It is formally defined in Table 4.

Table 4 – AnalogUnitType definition

Attribute	**Value**				
BrowseName	AnalogUnitType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseAnalogType* defined in 5.3.2.2, i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.2.5 AnalogUnitRangeType

The *AnalogUnitRangeType* derives from the *AnalogItemType* and additionaly requires the *EngineeringUnits Property*. It is formally defined in Table 5.

Table 5 – AnalogUnitRangeType definition

Attribute	**Value**				
BrowseName	AnalogUnitRangeType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalogItemType* defined in 5.3.2.3, i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.3 DiscreteItemType

5.3.3.1 General

This VariableType is an abstract type. That is, no instances of this type can exist. However, it might be used in a filter when browsing or querying. The *DiscreteItemType* derives from the *DataItemType* and therefore shares all of its characteristics. It is formally defined in Table 6.

Table 6 – DiscreteItemType definition

Attribute	**Value**				
BrowseName	DiscreteItemType				
IsAbstract	True				
ValueRank	−2 (−2 = ‘Any’)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.2; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	TwoStateDiscreteType	Defined in 5.3.3.2		
HasSubtype	VariableType	MultiStateDiscreteType	Defined in 5.3.3.3		
HasSubtype	VariableType	MultiStateValueDiscreteType	Defined in 5.3.3.4		

5.3.3.2 TwoStateDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have two states. The *TwoStateDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 7.

Table 7 – TwoStateDiscreteType definition

Attribute	Value				
BrowseName	TwoStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Boolean				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	TrueState	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	FalseState	LocalizedText	PropertyType	Mandatory

TrueState contains a string to be associated with this *DataItem* when it is TRUE. This is typically used for a contact when it is in the closed (non-zero) state.

for example: "RUN", "CLOSE", "ENABLE", "SAFE", etc.

FalseState contains a string to be associated with this *DataItem* when it is FALSE. This is typically used for a contact when it is in the open (zero) state.

for example: "STOP", "OPEN", "DISABLE", "UNSAFE", etc.

If the item contains an array, then the *Properties* will apply to all elements in the array.

The *StatusCode SemanticsChanged* bit shall be set if any of the *FalseState or TrueState (*changes can cause misinterpretation by users or (scripting) programs*) Properties* are changed (see 5.2 for additional information).

5.3.3.3 MultiStateDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have more than two states. The *MultiStateDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 8.

Table 8 – MultiStateDiscreteType definition

Attribute	Value				
BrowseName	MultiStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	UInteger				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EnumStrings	LocalizedText[]	PropertyType	Mandatory

EnumStrings is a string lookup table corresponding to sequential numeric values (0, 1, 2, etc.)

Example:

"OPEN"
"CLOSE"
"IN TRANSIT" etc.

Here the string "OPEN" corresponds to 0, "CLOSE" to 1 and "IN TRANSIT" to 2.

Clients should be prepared to handle item values outside of the range of the list; and robust servers should be prepared to handle writes of illegal values.

If the item contains an array, then this lookup table shall apply to all elements in the array.

NOTE The *EnumStrings* property is also used for Enumeration *DataTypes* (for the specification of this *DataType*, see IEC 62541-3).

The *StatusCode SemanticsChanged* bit shall be set if the *EnumStrings (*changes can cause misinterpretation by users or (scripting) programs*) Property* is changed (see 5.2 for additional information).

5.3.3.4 MultiStateValueDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have more than two states and where the state values (the enumeration) does not consist of consecutive numeric values (may have gaps) or where the enumeration is not zero-based. The *MultiStateValueDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 9.

Table 9 – MultiStateValueDiscreteType definition

Attribute	**Value**				
BrowseName	MultiStateValueDiscreteType				
IsAbstract	False				
ValueRank	Scalar				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EnumValues	See IEC 62541-3		Mandatory
HasProperty	Variable	ValueAsText	See IEC 62541-3		Mandatory

EnumValues is an array of *EnumValueType*. Each entry of the array represents one enumeration value with its integer notation, a human-readable representation, and help information. This represents enumerations with integers that are not zero-based or have gaps (e.g. 1, 2, 4, 8, 16). See IEC 62541-3 for the definition of this type. *MultiStateValueDiscrete Variables* expose the current integer notation in their *Value Attribute*. *Clients* will often read the *EnumValues Property* in advance and cache it to lookup a name or help whenever they receive the numeric representation.

~~*MultiStateValueDiscrete Variables* can have any numeric *Data Type*; this includes signed and unsigned integers from 8 to 64 Bit length.~~

Only *DataTypes* that can be represented with *EnumValues* are allowed for *Variables* of *MultiStateValueDiscreteType*. These are*:*

- signed integers up to 64 bits in length;
- unsigned integers up to 63 bits in length.

The numeric representation of the current enumeration value is provided via the *Value Attribute* of the *MultiStateValueDiscrete Variable*. The *ValueAsText Property* provides the localized text representation of the enumeration value. It can be used by *Clients* only interested in displaying the text to subscribe to the *Property* instead of the *Value Attribute*.

5.3.4 ArrayItemType

5.3.4.1 General

This abstract *VariableType* defines the general characteristics of an *ArrayItem*. Values are exposed in an array, but the content of the array represents a single entity like an image. Other *DataItems* might contain arrays that represent for example several values of several temperature sensors of a boiler.

ArrayItemType or its subtype shall only be used when the *Title* and *AxisScaleType Properties* can be filled with reasonable values. If this is not the case *DataItemType* and subtypes like *AnalogItemType,* which also support arrays, shall be used. The *ArrayItemType* is formally defined in Table 10.

Table 10 – ArrayItemType definition

Attribute	**Value**				
BrowseName	ArrayItemType				
IsAbstract	True				
ValueRank	0 (0 = OneOrMoreDimensions)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.3.1; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	YArrayItemType	Defined in 5.3.4.2		
HasSubtype	VariableType	XYArrayItemType	Defined in 5.3.4.3		
HasSubtype	VariableType	ImageItemType	Defined in 5.3.4.4		
HasSubtype	VariableType	CubeItemType	Defined in 5.3.4.5		
HasSubtype	VariableType	NDimensionArrayItemType	Defined in 5.3.4.6		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Mandatory
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory
HasProperty	Variable	Title	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	AxisScaleType	AxisScaleEnumeration	PropertyType	Mandatory

InstrumentRange defines the range of the *Value* of the *ArrayItem.*

EURange defines the value range of the ArrayItem likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.

EngineeringUnits holds the information about the engineering units of the *Value* of the *ArrayItem*.

For additional information about *InstrumentRange*, *EURange*, and *EngineeringUnits* see the description of *AnalogItemType* in 5.3.2.

Title holds the user readable title of the *Value* of the *ArrayItem*.

AxisScaleType defines the scale to be used for the axis where the *Value* of the *ArrayItem* shall be displayed.

The *StatusCode SemanticsChanged* bit shall be set if any of the *InstrumentRange*, *EURange*, *EngineeringUnits* or *Title Properties* are changed (see 5.2 for additional information).

5.3.4.2 YArrayItemType

YArrayItemType represents a single-dimensional array of numerical values used to represent spectra or distributions where the x axis intervals are constant. *YArrayItemType* is formally defined in Table 11.

Table 11 – YArrayItemType definition

Attribute	Value				
BrowseName	YArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	BaseDataType				
ArrayDimensions	{0} (0 = UnknownSize)				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

The *Value* of the *YArrayItem* contains the numerical values for the Y-Axis. *Engineering Units* and *Range* for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* and *DoubleComplexNumberType*.

The *XAxisDefinition Property* holds the information about the *Engineering Units* and *Range* for the X-Axis.

The *StatusCode SemanticsChanged* bit shall be set if any of the following five *Properties* are changed: *InstrumentRange, EURange, EngineeringUnits, Title* or *XAxisDefinition* (see 5.2 for additional information).

Figure 3 shows an example of how *Attributes* and *Properties* may be used in a graphical interface.

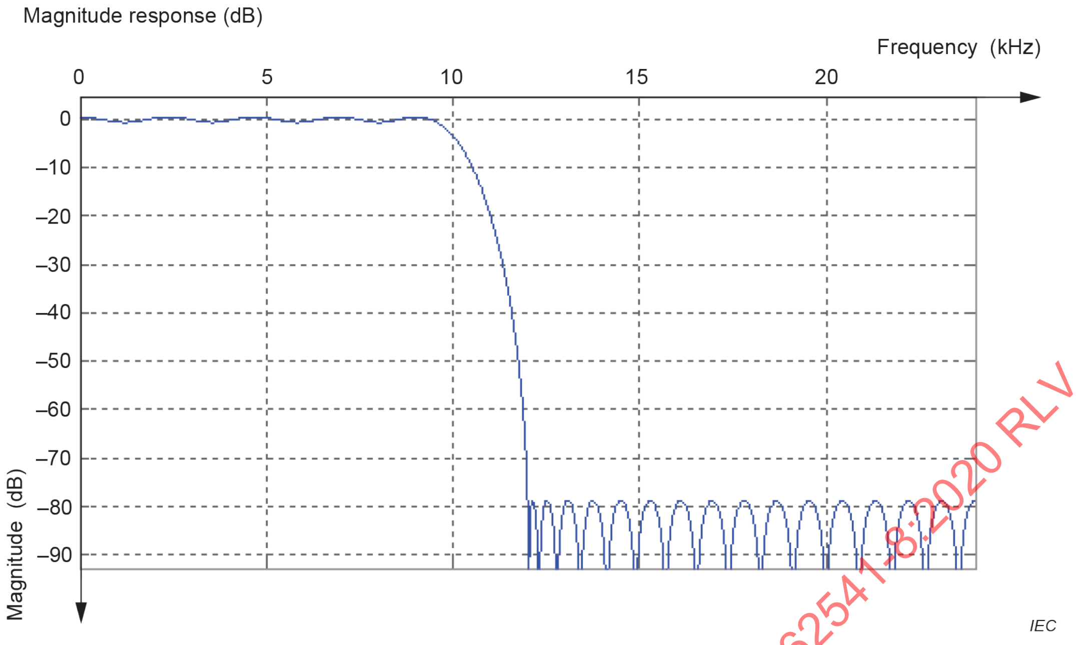


Figure 3 – Graphical view of a *YArrayItem*

Table 12 describes the values of each element presented in Figure 3.

Table 12 – *YArrayItem* item description

Attribute / Property	Item value
Description	Magnitude Response (dB)
axisScaleType	AxisScaleEnumeration.LINEAR_0
InstrumentRange.low	-90
InstrumentRange.high	5
EURange.low	-90
EURange.high	2
EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
EngineeringUnits.unitId	2N
EngineeringUnits.displayName	"en-us", "dB"
EngineeringUnits.description	"en-us", "decibel"
Title	Magnitude
XAxisDefinition.EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
XAxisDefinition.EngineeringUnits.unitId	kHz
XAxisDefinition.EngineeringUnits.displayName	"en-us", "kHz"
XAxisDefinition.EngineeringUnits.description	"en-us", "kilohertz"
XAxisDefinition.Range.low	0
XAxisDefinition.Range.high	25
XAxisDefinition.title	"en-us", "Frequency"
XAxisDefinition.axisScaleType	AxisScaleEnumeration.LINEAR_0
XAxisDefinition.axisSteps	null
Interpretation notes: • Not all elements of this table are used in Figure 3. • The X axis is displayed in reverse order, however, the *XAxisDefinition.Range.low* shall be lower than *XAxisDefinition.Range.high*. It is only a graphical representation that reverses the display order. • There is a constant X axis.	

5.3.4.3 XYArrayItemType

XYArrayItemType represents a vector of XVType values like a list of peaks, where XVType.x is the position of the peak and XVType.value is its intensity. *XYArrayItemType* is formally defined in Table 13.

Table 13 – XYArrayItemType definition

Attribute	Value				
BrowseName	XYArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	XVType (defined in 5.6.8)				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

The *Value* of the X*YArrayItem* contains an array of structures (XVType) where each structure specifies the position for the X-Axis (XVType.x) and the value itself (XVType.value), used for the Y-Axis. Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

XAxisDefinition Property holds the information about the *Engineering Units* and *Range* for the X-Axis.

The *axisSteps* of *XAxisDefinition* shall be set to NULL because it is not used.

The *StatusCode SemanticsChanged* bit shall be set if any of the *InstrumentRange, EURange, EngineeringUnits, Title* or *XAxisDefinition Properties* are changed (see 5.2 for additional information).

5.3.4.4 ImageItemType

ImageItemType defines the general characteristics of an ImageItem which represents a matrix of values like an image, where the pixel position is given by X which is the column and Y the row. The value is the pixel intensity.

ImageItemType is formally defined in Table 14.

Table 14 – ImageItemType definition

Attribute	**Value**				
BrowseName	ImageItemType				
IsAbstract	False				
ValueRank	2 (2 = two dimensional array)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, ComplexNumberType and DoubleComplexNumberType.

The *ArrayDimensions Attribute* for *Variables* of this type or subtypes shall use the first entry in the array ([0]) to define the number of columns and the second entry ([1]) to define the number of rows, assuming the size of the matrix is not dynamic.

XAxisDefinition Property holds the information about the engineering units and range for the X-Axis.

YAxisDefinition Property holds the information about the engineering units and range for the Y-Axis.

The *StatusCode.SemanticsChanged* bit shall be set if any of the *InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition* or *YAxisDefinition Properties* are changed.

5.3.4.5 CubeItemType

CubeItemType represents a cube of values like a spatial particle distribution, where the particle position is given by X which is the column, Y the row and Z the depth. In the example of a spatial partical distribution, the value is the particle size. *CubeItemType* is formally defined in Table 15.

Table 15 – CubeItemType definition

Attribute	**Value**				
BrowseName	CubeItemType				
IsAbstract	False				
ValueRank	3 (3 = three dimensional array)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	ZAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* and *DoubleComplexNumberType*.

The *ArrayDimensions Attribute* for *Variables* of this type or subtypes should use the first entry in the array ([0]) to define the number of columns, the second entry ([1]) to define the number of rows, and the third entry ([2]) define the number of steps in the Z axis, assuming the size of the matrix is not dynamic.

XAxisDefinition Property holds the information about the engineering units and range for the X-Axis.

YAxisDefinition Property holds the information about the engineering units and range for the Y-Axis.

ZAxisDefinition Property holds the information about the engineering units and range for the Z-Axis.

The *StatusCode SemanticsChanged* bit shall be set if any of *the InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition, YAxisDefinition* or *ZAxisDefinition Properties* are changed (see 5.2 for additional information).

5.3.4.6 NDimensionArrayItemType

This *VariableType* defines a generic multi-dimensional *ArrayItem*.

This approach minimizes the number of types however it may be proved more difficult to utilize for control system interactions.

NDimensionArrayItemType is formally defined in Table 16.

Table 16 – NDimensionArrayItemType definition

Attribute	Value				
BrowseName	NdimensionArrayItemType				
IsAbstract	False				
ValueRank	0 (0 = OneOrMoreDimensions)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	AxisDefinition	AxisInformation []	PropertyType	Mandatory

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, ComplexNumberType and DoubleComplexNumberType.

AxisDefinition Property holds the information about the *Engineering Units* and *Range* for all axis.

The *StatusCode SemanticsChanged* bit shall be set if any of *the InstrumentRange, EURange, EngineeringUnits, Title* or *AxisDefinition Properties* are changed (see 5.2 for additional information).

5.4 Address Space model

*DataItem*s are always defined as data components of other *Node*s in the *AddressSpace*. They are never defined by themselves. A simple example of a container for *DataItem*s would be a "Folder Object" but it can be an *Object* of any other type.

Figure 4 illustrates the basic *AddressSpace* model of a *DataItem*, in this case an *AnalogItem*.

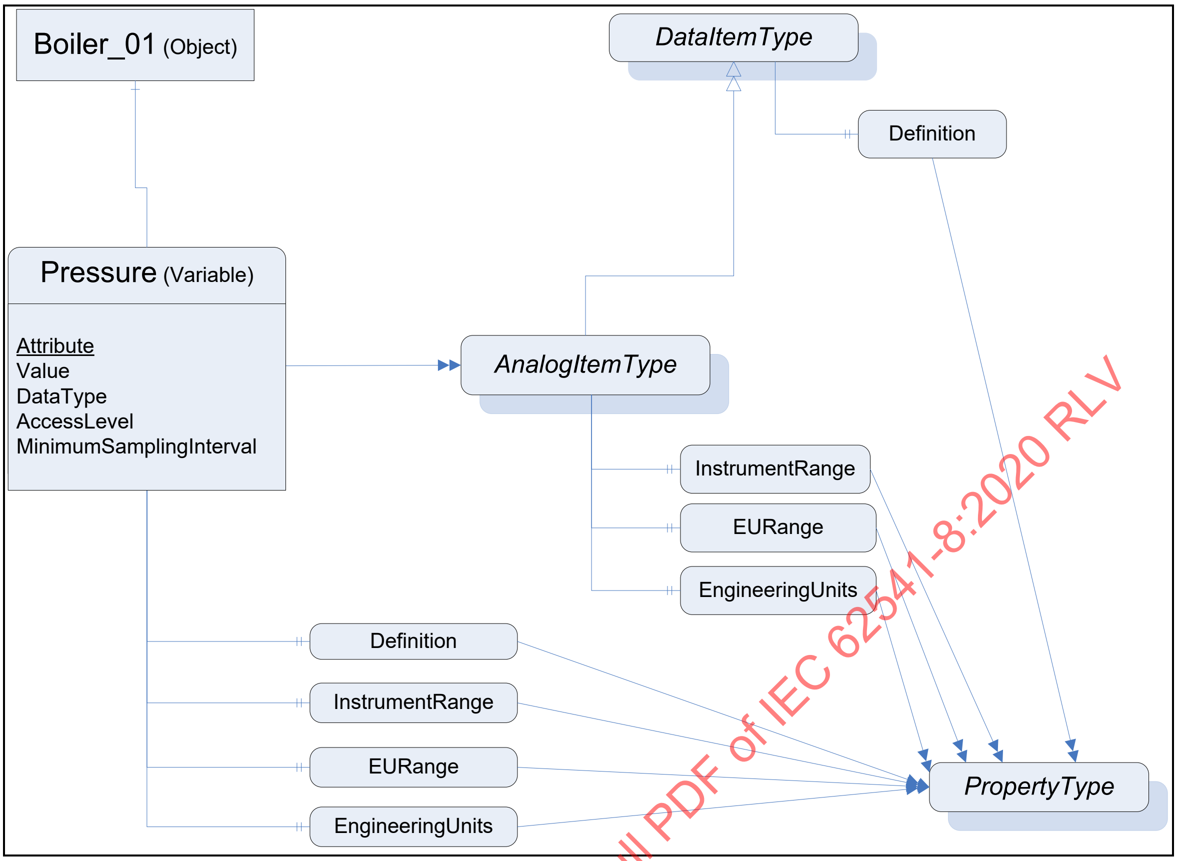


Figure 4 – Representation of DataItems in the AddressSpace

Each *DataItem* is represented by a *DataVariable* with a specific set of *Attributes*. The *TypeDefinition* reference indicates the type of the *DataItem* (in this case the *AnalogItemType*). Additional characteristics of *DataItems* are defined using *Properties*. The *VariableTypes* in 5.2 specify which properties may exist. These *Properties* have been found to be useful for a wide range of Data Access clients. *Servers* that want to disclose similar information should use the OPC-defined *Property* rather than one that is vendor-specific.

The above figure shows only a subset of *Attributes* and *Properties*. Other *Attributes* that are defined for *Variables* in IEC 62541-3 (e.g., *Description*) may also be available.

5.5 Attributes of DataItems

This subclause lists the *Attributes* of *Variables* that have particular importance for Data Access. They are specified in detail in IEC 62541-3. The following *Attributes* are particularly important for Data Access:

- Value,
- DataType,
- AccessLevel,
- MinimumSamplingInterval.

Value is the most recent value of the *Variable* that the *Server* has. Its data type is defined by the *DataType Attribute*. The *AccessLevel Attribute* defines the *Server's* basic ability to access current data and *MinimumSamplingInterval* defines how current the data is.

When a client requests the *Value Attribute* for reading or monitoring, the *Server* will always return a *StatusCode* (the quality and the *Server's* ability to access/provide the value) and, optionally, a *ServerTimestamp* and/or a *SourceTimestamp* – based on the *Client's* request. See IEC 62541-4 for details on *StatusCode* and the meaning of the two timestamps. Specific status codes for Data Access are defined in 6.3.

5.6 DataTypes

5.6.1 Overview

Following is a description of the *DataTypes* defined in this specification.

DataTypes like *String, Boolean*, *Double* or *LocalizedText* are defined in IEC 62541-3. Their representation is specified in IEC 62541-5.

5.6.2 Range

This structure defines the *Range* for a value. Its elements are defined in Table 17.

Table 17 – *Range* DataType structure

Name	Type	Description
Range	structure	
low	Double	Lowest value in the range.
high	Double	Highest value in the range.

If the *DataType* of the *Variable* is Int64 or UInt64 not all values can be covered with the *Range DataType*. In that case, the next lowest respectively the next highest value shall be used.

If a limit is not known a NaN shall be used.

Its representation in the *AddressSpace* is defined in Table 18.

Table 18 – *Range* definition

Attributes	Value
BrowseName	Range

5.6.3 EUInformation

This structure contains information about the *EngineeringUnits*. Its elements are defined in Table 19.

Table 19 – *EUInformation* DataType structure

Name	Type	Description
EUInformation	structure	
namespaceUri	String	Identifies the organization (company, standards organization) that defines the *EUInformation*.
unitId	Int32	Identifier for programmatic evaluation. −1 is used if a *unitId* is not available.
displayName	LocalizedText	The *displayName* of the engineering unit is typically the abbreviation of the engineering unit, for example "h" for hour or "m/s" for meter per second.
description	LocalizedText	Contains the full name of the engineering unit such as "hour" or "meter per second".

Its representation in the *AddressSpace* is defined in Table 20.

Table 20 – *EUInformation* definition

Attributes	Value
BrowseName	EUInformation

To facilitate interoperability, OPC UA specifies how to apply the widely accepted "Codes for Units of Measurement" published by the "United Nations Centre for Trade Facilitation and Electronic Business" (see UN/CEFACT: UNECE Recommendation N° 20). It uses and is based on the International System of Units (SI Units) but in addition provides a fixed code that can be used for automated evaluation. This recommendation has been accepted by many industries on a global basis.

The UNECE recommendation can be found here:

https://www.unece.org/cefact/codesfortrade/codes_index.html

The latest UNECE version (Rev 12. Filename = rec20_Rev12e_2016.xls, published in 2016) is available here:

http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_Rev12e_2016.xls

The mapping of the UNECE codes to OPC UA (EUInformation.unitId) is available here:

http://www.opcfoundation.org/UA/EngineeringUnits/UNECE/UNECE_to_OPCUA.csv

Table 21 contains a small excerpt of the published Annex with Code Lists:

Table 21 – Examples from UNECE Recommendation N° 20

Excerpt from Recommendation N°. 20, Annex 1			
Common Code	**Name**	**Conversion Factor**	**Symbol**
C81	radian		rad
C25	milliradian	10^{-3} rad	mrad
MMT	millimetre	10^{-3} m	mm
HMT	hectometre	10^{2} m	hm
KTM	kilometre	10^{3} m	km
KMQ	kilogram per cubic metre	kg/m^3	kg/m^3
FAH	degree Fahrenheit	5/9 × K	° F
J23	degree Fahrenheit per hour	1,543 210 × 10^{-4} K/s	° F/h

Specific columns of this table shall be used to create the *EUInformation* structure as defined by the following rules:

- The Common Code is represented as an alphanumeric variable length of 3 characters. It shall be used for the *EUInformation.unitId*. The following pseudo code specifies the algorithm to convert the Common Code into an Int32 as needed for *EUInformation.unitId*:

```
Int32 unitId = 0;
Int32 c;
for (i=0; i<=3;i++)
{
    c = CommonCode[i];
    if (c == 0) break;          // end of Common Code
    unitId = unitId << 8;
    unitId = unitId | c;
}
```

- The Symbol field shall be copied to the *EUInformation.displayName*. The localeId field of *EUInformation.displayName* shall be empty.
- The Name field shall be used for *EUInformation.description*. If the name is copied, then the localeId field of *EUInformation.description* shall be empty. If the name is localized, then the localeId field shall specify the correct locale.

The *EUInformation.namespaceUri* shall be http://www.opcfoundation.org/UA/units/un/cefact.

NOTE It ~~will be~~ is advantegous to use Recommendation N° 20 as specified, because it can be programmatically interpreted by generic OPC UA *Clients*. However, the *EUInformation* structure has been defined such that other standards bodies can incorporate their engineering unit definitions into OPC UA. If *Servers* use such an approach, then they shall identify this standards body by using a proper *namespaceUri* in *EUInformation.namespaceUri*.

5.6.4 ComplexNumberType

This structure defines float IEEE 32 bits complex value. Its elements are defined in Table 22.

Table 22 – ComplexNumberType DataType structure

Name	Type	Description
ComplexNumberType	structure	
real	Float	Value real part
imaginary	Float	Value imaginary part

Its representation in the *AddressSpace* is defined in Table 23.

Table 23 – ComplexNumberType definition

Attributes	Value
BrowseName	ComplexNumberType

5.6.5 DoubleComplexNumberType

This structure defines double IEEE 64 bits complex value. Its elements are defined in Table 24.

Table 24 – DoubleComplexNumberType DataType structure

Name	Type	Description
DoubleComplexNumberType	structure	
real	Double	Value real part
imaginary	Double	Value imaginary part

Its representation in the *AddressSpace* is defined in Table 25.

Table 25 – DoubleComplexNumberType definition

Attributes	Value
BrowseName	DoubleComplexNumberType

5.6.6 AxisInformation

This structure defines the information for auxiliary axis for *ArrayItemType Variable*s.

There are three typical uses of this structure:

a) the step between points is constant and can be predicted using the range information and the number of points. In this case, *axisSteps* can be set to NULL;

a) the step between points is not constant, but remains the same for a long period of time (from acquisition to acquisition for example). In this case, *axisSteps* contains the value of each step on the axis;

b) the step between points is not constant and changes at every update. In this case, a type like *XYArrayType* shall be used and *axisSteps* is set to NULL.

Its elements are defined in Table 26.

Table 26 – AxisInformation DataType structure

Name	Type	Description
AxisInformation	structure	
engineeringUnits	EUInformation	Holds the information about the engineering units for a given axis.
eURange	Range	Limits of the range of the axis
title	Localizedtext	User readable axis title, useful when the units are %, the Title may be "Particle size distribution"
axisScaleType	AxisScaleEnumeration	LINEAR, LOG, LN, defined by AxisSteps
axisSteps	Double[]	Specific value of each axis steps, may be set to "Null" if not used

When the steps in the axis are constant, *axisSteps* may be set to "Null" and in this case, the *Range* limits are used to compute the steps. The number of steps in the axis comes from the parent *ArrayItem.ArrayDimensions*.

5.6.7 AxisScaleEnumeration

This enumeration identifies on which type of axis the data shall be displayed. Its values are defined in Table 27.

Table 27 – AxisScaleEnumeration values

Value	Description
LINEAR_0	Linear scale
LOG_1	Log base 10 scale
LN_2	Log base e scale

Its representation in the *AddressSpace* is defined in Table 28.

Table 28 – AxisScaleEnumeration definition

Attributes	Value
BrowseName	AxisScaleEnumeration

5.6.8 XVType

This structure defines a physical value relative to a X axis and it is used as the *DataType* of the Value of *XYArrayItemType*. For details see 5.3.4.3.

Many devices can produce values that can perfectly be represented with a float IEEE 32 bits but, they can position them on the X axis with an accuracy that requires double IEEE 64 bits. For example, the peak value in an absorbance spectrum where the amplitude of the peak can be represented by a float IEEE 32 bits, but its frequency position required 10 digits which implies the use of a double IEEE 64 bits.

Its elements are defined in Table 29.

Table 29 – XVType DataType structure

Name	Type	Description
XVType	structure	
x	Double	Position on the X axis of this value
value	Float	The value itself

Its representation in the *AddressSpace* is defined in Table 30.

Table 30 – XVType definition

Attributes	Value
BrowseName	XVType

6 Data Access specific usage of Services

6.1 General

IEC 62541-4 specifies the complete set of services. The services needed for the purpose of DataAccess are:

- The *View* service set and *Query* service set to detect *DataItems*, and their *Properties*.
- The *Attribute* service set to read or write *Attributes* and in particular the value *Attribute*.
- The *MonitoredItem* and *Subscription* service set to set up monitoring of *DataItems* and to receive data change notifications.

6.2 PercentDeadband

The *DataChangeFilter* in IEC 62541-4 defines the conditions under which a data change notification shall be reported. This filter contains a *deadbandValue* which can be of type *AbsoluteDeadband* or *PercentDeadband*. IEC 62541-4 already specifies the behaviour of the *AbsoluteDeadband*. This sub-clause specifies the behaviour of the *PercentDeadband* type.

DeadbandType = PercentDeadband

For this type of deadband the *deadbandValue* is defined as the percentage of the *EURange*. That is, it applies only to *AnalogItems* with an *EURange Property* that defines the typical value range for the item. This range shall be multiplied with the *deadbandValue* and then compared to the actual value change to determine the need for a data change notification. The following pseudo code shows how the deadband is calculated:

```
DataChange if (absolute value of (last cached value - current value) >
                          (deadbandValue/100.0) * ((high-low) of EURange)))
```

The range of the *deadbandValue* is from 0,0 to 100,0 per cent. Specifying a *deadbandValue* outside of this range will be rejected and reported with the *StatusCode* Bad_DeadbandFilterInvalid (see Table 31).

If the Value of the *MonitoredItem* is an array, then the deadband calculation logic shall be applied to each element of the array. If an element that requires a DataChange is found, then no further deadband checking is necessary and the entire array shall be returned.

6.3 Data Access status codes

6.3.1 Overview

This subclause defines additional codes and rules that apply to the *StatusCode* when used for Data Access values.

The general structure of the *StatusCode* is specified in IEC 62541-4 and includes a set of common operational result codes that also apply to Data Access.

6.3.2 Operation level result codes

Certain conditions under which a *Variable* value was generated are only valid for automation data and in particular for device data; they are similar, but are slightly more generic than the description of data quality in the various fieldbus specifications.

Table 31 contains codes with BAD severity which indicates a failure.

Table 32 contains codes with UNCERTAIN severity which indicates that the value has been generated under sub-normal conditions.

Table 33 contains GOOD (success) codes.

Note again, that these are the codes that are specific for Data Access and supplement the codes that apply to all types of data which are defined in IEC 62541-4.

Table 31 – Operation level result codes for BAD data quality

Symbolic Id	Description
~~**Remarks:**~~	
Bad is defined in IEC 62541-4. It shall be used when there is no special reason why the Value is bad.	

Bad_ConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
Bad_NotConnected	The variable should receive its value from ~~another variable~~ some data source, but has never been configured to do so.
Bad_DeviceFailure	There has been a failure in the device/data source that generates the value that has affected the value.
Bad_SensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source. The limits bits are used to define if the limits of the value have been reached.
~~Remarks:~~ Bad_NoCommunication is defined in IEC 62541-4. It shall be used when communications to the data source is defined, but not established, and there is no last known value available.	
Bad_OutOfService	The source of the data is not operational.
~~Bad_LastKnown~~	OPC UA requires that the *Server* shall return a Null value when the *Severity* is Bad. Therefore, the Fieldbus code "Bad_LastKnown" shall be mapped to Uncertain_NoCommunicationLastUsable.
Bad_DeadbandFilterInvalid	The specified *PercentDeadband* is not between 0.0 and 100.0 or a *PercentDeadband* is not supported, since an *EURange* is not configured.
~~Remarks:~~ Bad_WaitingForInitialData is defined in IEC 62541-4.	

Table 32 – Operation level result codes for UNCERTAIN data quality

Symbolic Id	Description
~~Remarks:~~ **Uncertain** is defined in IEC 62541-4. It shall be used when there is no special reason why the Value is uncertain.	
Uncertain_NoCommunicationLastUsable	Communication to the data source has failed. The variable value is the last value that had a good quality and it is uncertain whether this value is still current. The server timestamp in this case is the last time that the communication status was checked. The time at which the value was last verified to be true is no longer available.
Uncertain_LastUsableValue	Whatever was updating this value has stopped doing so. This happens when an input variable is configured to receive its value from another variable and this configuration is cleared after one or more values have been received. This status/substatus is not used to indicate that a value is stale. Stale data can be detected by the client looking at the timestamps.
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. This status/substatus is set only during configuration while the variable is not operational (while it is out-of-service).
Uncertain_SensorNotAccurate	The value is at one of the sensor limits. The Limits bits define which limit has been reached. Also set if the device can determine that the sensor has reduced accuracy (e.g. degraded analyzer), in which case the Limits bits indicate that the value is not limited.
Uncertain_EngineeringUnitsExceeded	The value is outside of the range of values defined for this parameter. The Limits bits indicate which limit has been reached or exceeded.
Uncertain_SubNormal	The value is derived from multiple sources and has less than the required number of Good sources.

Table 33 – Operation level result codes for GOOD data quality

Symbolic Id	Description
~~Remarks:~~ Good is defined in IEC 62541-4. It shall be used when there are no special conditions.	
Good_LocalOverride	The value has been Overridden. Typically, this means the input has been disconnected and a manually-entered value has been "forced".

6.3.3 LimitBits

The bottom 16 bits of the *StatusCode* are bit flags that contain additional information, but do not affect the meaning of the *StatusCode*. Of particular interest for *DataItem*s is the *LimitBits* field. In some cases, such as sensor failure it can provide useful diagnostic information.

Servers that do not support Limit have to set this field to 0.

Annex A
(informative)

OPC COM DA to UA mapping

A.1 Overview

This Annex provides details on mapping OPC COM Data Access (DA) information to OPC UA to help vendors migrate to OPC UA based systems while still being able to access information from existing OPC COM DA systems.

The OPC Foundation provides COM UA Wrapper and Proxy samples that act as a bridge between the OPC DA and the OPC UA systems.

The COM UA Wrapper is an OPC UA Server that wraps an OPC DA Server and with that enables an OPC UA Client to access information from the DA Server. The COM UA Proxy enables an OPC DA Client to access information from an OPC UA Server.

The mappings describe generic DA interoperability components. It is recommended that vendors use this mapping if they develop their own components, however, some applications may benefit from vendor-specific mappings.

A.2 Security considerations

COM DA relies on the Microsoft COM security infrastructure and does not specify any security parameters such as user identity. The developer of UA Wrapper and Proxy therefore has to consider the mapping of security aspects.

The COM UA Wrapper for instance may accept any Username/password and then try to impersonate this user by calling proper Windows services before connecting to the COM DA Server.

A.3 COM UA wrapper for OPC DA Server

A.3.1 Information Model mapping

A.3.1.1 General

OPC DA defines 3 elements in the address space: Branch, Item and Property. The COM UA Wrapper maps these types to the OPC UA types as described in A.3.1.2 to A.3.1.4.

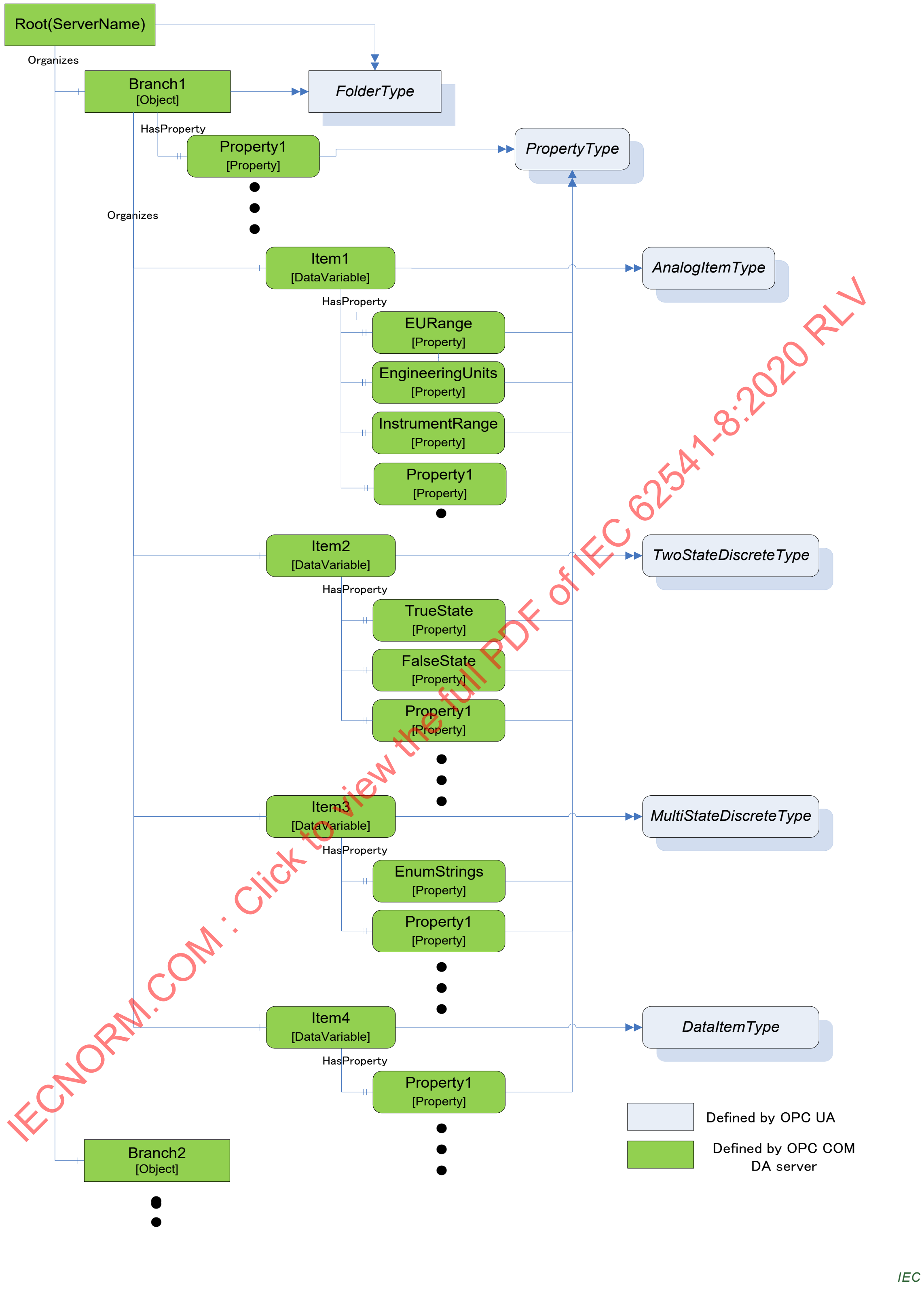


Figure A.1 – Sample OPC UA Information Model for OPC DA

A.3.1.2 Branch

DA Branches are represented in the COM UA Wrapper as *Objects* of *FolderType*.

The top-level branch (the root) should be represented by an *Object* where the *BrowseName* is the Server ProgId.

The OPC DA Address space hierarchy is discovered using the ChangeBrowsePosition from the Root and BrowseOPCItemIds to get the Branches, Items and Properties.

The name returned from the BrowseOPCItemIds enumString is used as the BrowseName and the DisplayName for each Branch. See also A.3.1.5.

The ItemId obtained using the GetItemID is used as a part of the NodeId for each Branch. See also A.3.1.5.

An OPC UA *Folder* representing a DA Branch uses the *Organizes References* to reference child DA Branches and uses *HasComponent References* for DA Leafs (Items). It is acceptable for customized wrappers to use a sub-type of these ReferenceTypes.

A.3.1.3 Item

DA items (leafs) are represented in the COM UA Wrapper as *Variables*. The VariableType depends on the existance of special DA properties as follows:

- *AnalogItemType*: An item in the DA server that has High EU and Low EU properties or its EU Type property is Analog is represented as *Variable* of *AnalogItemType* in the COM UA Wrapper. *The AnalogItemType* has the following *Properties:*
  - *EURange*: The values of the High EU and Low EU properties of the DA Item are assigned to the *EURange Property*
  - *EngineeringUnits*: The value of the Engineering Unit property of the DA Item are assigned to the *EngineeringUnits Property*.
  - InstrumentRange: The values of the High IR and Low IR properties of the DA Item are assigned to the InstrumentRange Property
- *TwoStateDiscreteType: An* item in DA server that has Open Label and Close Label properties is represented as *Variable* of *TwoStateDiscreteType* in the COM UA Wrapper. *The TwoStateDiscreteType* has the following *Properties*
  - *TrueState*: The value of the Close Label property of the DA item is assigned to the *TrueState Property*.
  - FalseState: The value of the Open Label property of the DA item is assigned to the FalseState Property.
- *MultiStateDiscreteType***:** An item in the DA server that has its EU Type property as enumerated is represented as *Variable* of *MultiStateDiscreteType* in the COM UA Wrapper. The *MultiStateDiscreteType* has the following *Property*:
  - *EnumStrings*: The enumerated values of the EUInfo Property of the DA item are assigned to the *EnumStrings Property*.
- *DataItemType***:** An item in the DA Server that is not any of the above types is represented as *Variable* of *DataItemType* in the COM UA Wrapper*.*

Below are mappings that are common for all item types

- The name of the item in the DA Server is used as the *BrowseName* and the *DisplayName* for the *Node* in the COM UA Wrapper. See also clause A.3.1.5.
- The ItemId in the DA server is used as a part of the *NodeId* for the *Node*. See also clause A.3.1.5.
- TimeZone property in the DA server is represented by a *TimeZone Property.*
- The Description property value in the DA server is assigned to the *Description Attribute*.
- *The* DataType property value in the DA server is assigned *to the DataType Attribute*.
- If the item in the DA server is an array, the *ValueRank Attribute* is set as *OneOrMoreDimensions.* If not, it is set to *Scalar.*
- The *AccessLevel Attribute* is set with the AccessRights value in the DA server:

- OPC_READABLE -> Readable
- OPC_WRITABLE -> Writable

Note that the same values are also set for the UserAccessLevel in the COM UA Wrapper.

- *The* ScanRate property value in the DA server is assigned to the *MinimumSamplingInterval Attribute*.

Any *Properties* added to a Node in the COM UA Wrapper are referenced using the *HasProperty ReferenceType*.

A.3.1.4 Property

A property in the DA server is represented in the COM UA Wrapper as a *Variable* with *TypeDefinition* as *PropertyType*.

The properties for an item are retrieved using the QueryAvailableProperties call in the DA server.

Below are mappings of the property details to the OPC UA Property:

- The description of a property in the DA server is used as the *BrowseName* and the *DisplayName* of the Node in the COM UA Wrapper.
- The PropertyID and ItemID (if they exist for the property) in the DA server are used as a part of the *NodeID* for the node in the COM UA Wrapper.
- The DataType value in the DA server is used as value for the *DataType Attribute* of the *Property* in the COM UA Wrapper.
- If the property value in the DA server is an array, the *ValueRank Attribute* of the *Property* is set to *OneOrMoreDimensions.* Otherwise it is set to *Scalar.*
- If the property has an ItemID in the DA server, then the *AccessLevel* attribute for the Node is set to *ReadableOrWriteable*. If not, it is set to *Readable*.

Table A.1 shows the mapping between the common OPC COM DA properties to the OPC UA Node attributes/properties.

Table A.1 – OPC COM DA to OPC UA Properties mapping

Property Name (PropertyID) of OPC COM DA	OPC UA Information Model	OPC UA DataType
Access Rights (5)	AccessLevel Attribute	Int32
EU Units (100)	EngineeringUnits Property	String
Item Description (101)	Description Attribute	String
High EU (102)	EURange Property	Double
Low EU (103)	EURange Property	Double
High Instrument Range (104)	InstrumentRange Property	Double
Low Instrument Range (105)	InstrumentRange Property	Double
Close Label (106)	TrueState Property	String
Open Label (107)	FalseState Property	String
Other Properties (include Vendor specific Properties)	PropertyType	Based on the DataType of the Property

A.3.1.5 BrowseName and DisplayName Mapping

As described above, both the OPC UA Browsename and Displayname for Nodes representing COM DA Branches and Leafs are derived from the name of the corresponding item in the COM DA Server.

This name can only be acquired by using the COM DA Browse Services. In OPC UA, however, the BrowseName and DisplayName are Attributes that Clients can ask for at any time. There are several options to support this in a Wrapper but all of them have pros and cons. Here are some popular implementation options:

a) Allow browsing the complete COM DA Address Space and then build and persist an offline copy of it. Resolve the BrowseName by scanning this offline copy.
   - Pro: the ItemID can be used as is for the OPC UA NodeId.
   - Con: the initial browse can take a while and may have to be repeated for COM DA Servers with a dynamic Address Space.

b) Create OPC UA NodeId values that include both the COM DA ItemID and the Item name. When the OPC UA Client passes such a NodeId to read the BrowseName or DisplayName Attribute, the wrapper can easily extract the name from the NodeId value.
   - Pro: efficient and reliable.
   - Con: the NodeId will not represent the ItemId. It becomes difficult for human users to match the two IDs.

c) A number of COM DA Servers use ItemIDs that consist of a path where the path elements are separated with a delimiter and the last element is the item name. Wrappers may provide ways to configure the delimiter so that they can easily extract the item name.
   - Pro: efficient and reliable. The ItemID can be used as is for the OPC UA NodeId.
   - Con: not a generic solution. Only works for specific COM-DA Servers.

For wrappers that are custom to a specific Server, knowledge of the COM DA server address space can result in other optimizations or short cuts (i.e. the server will always have a certain schema/naming sequence, etc.).

A.3.2 Data and error mapping

A.3.2.1 General

In a DA server, Automation Data is represented by Value, Quality and Time Stamp for a Tag.

The COM UA Wrapper maps the VQT data to the Data Value and Diagnostic Info structures.

The Error codes returned by the DA server are based on the HRESULT type. The COM UA Wrapper maps this error code to an OPC UA Status Code. Figure A.2 illustrates this mapping.

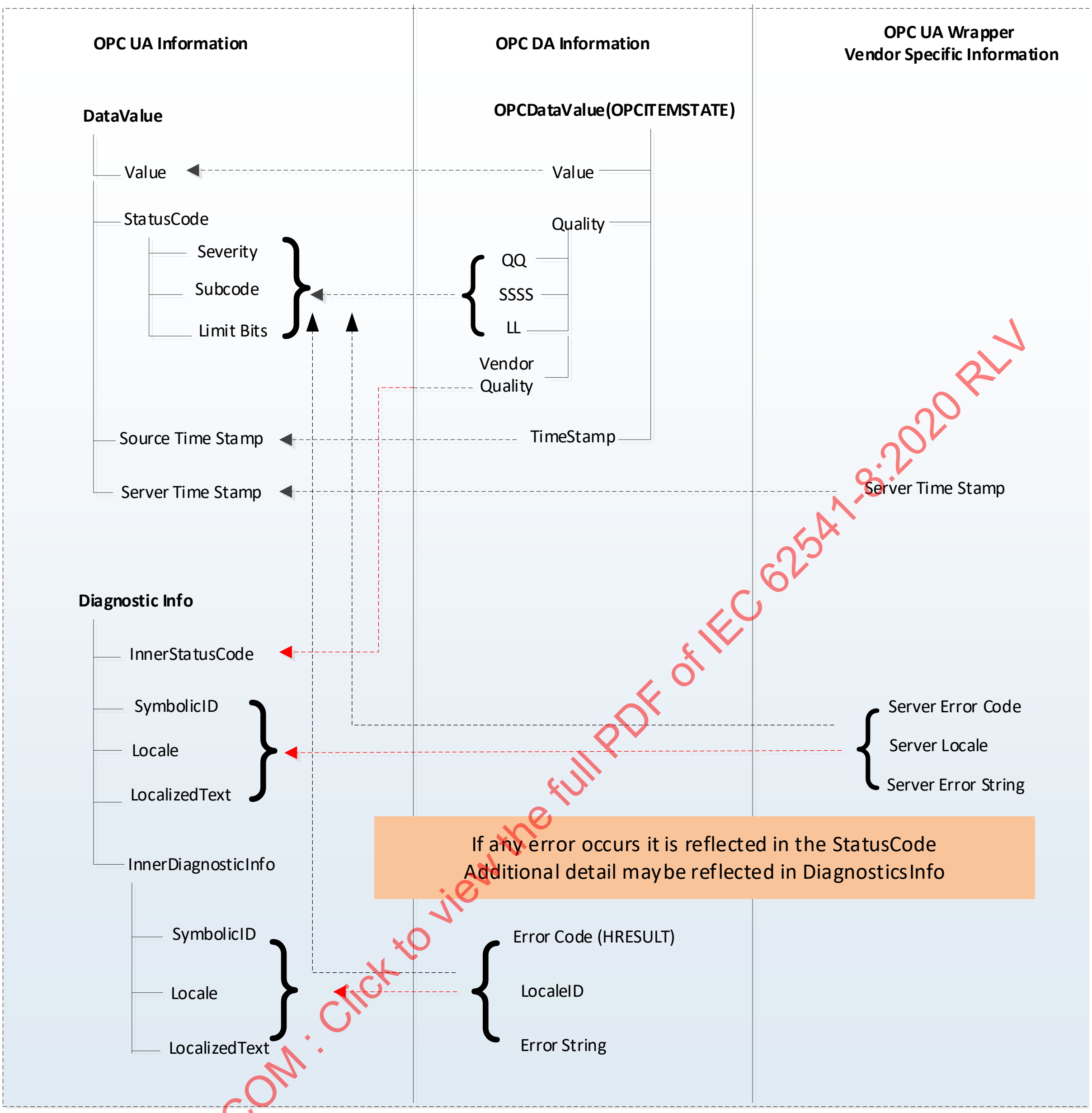


Figure A.2 – OPC COM DA to OPC UA data and error mapping

A.3.2.2 Value

The data values in the DA server are represented as Variant Data type. The COM UA Wrapper converts them to the corresponding OPC UA data type. The mapping is shown in Table A.2.

Table A.2 – DataTypes and mapping

Variant Data Type (In DA server)	OPC UA Data type Mapping in COM UA Server (DataValue structure)
VT_I2	Int16
VT_I4	Int32
VT_R4	Float
VT_R8	Double
VT_BSTR	String
VT_BOOL	Boolean
VT_UI1	Byte
VT_I1	SByte
VT_UI2	UInt16
VT_UI4	UInt32
VT_I8	Int64
VT_UI8	UInt64
VT_DATE	Double
VT_DECIMAL	Decimal
VT_ARRAY	Array of OPC UA types

A.3.2.3 Quality

The Quality of a Data Value in the DA server is represented as a 16-bit value where the lower 8 bits are of the form QQSSSSLL (Q: Main Quality, S: Sub Status, L: Limit) and the higher 8 bits are vendor specific.

The COM UA Wrapper maps the DA server to the OPC UA Status code as shown Figure A.3.

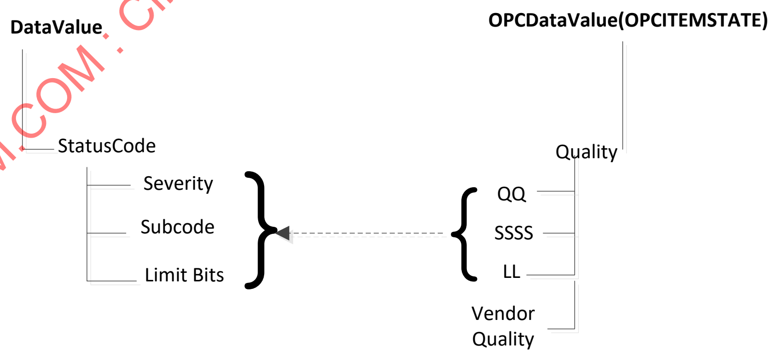


Figure A.3 – Status Code mapping

The primary quality is mapped to the Severity field of the Status code. The Sub Status is mapped to the SubCode and the Limit is mapped to the Limit Bits of the Status Code.

Please note that the Vendor quality is currently discarded.

Table A.3 shows a mapping of the OPC COM DA primary quality mapping to OPC UA status code

Table A.3 – Quality mapping

OPC DA Primary Quality (Quality & Sub status QQSSSS)	OPC UA Status Code
GOOD	Good
LOCAL_OVERRIDE	Good_LocalOverride
UNCERTAIN	Uncertain
SUB_NORMAL	Uncertain_SubNormal
SENSOR_CAL	Uncertain_SensorNotAccurate
EGU_EXCEEDED	Uncertain_EngineeringUnitsExceeded
LAST_USABLE	Uncertain_LastUsableValue
BAD	Bad
CONFIG_ERROR	Bad_ConfigurationError
NOT_CONNECTED	Bad_NotConnected
COMM_FAILURE	Bad_NoCommunication
DEVICE_FAILURE	Bad_DeviceFailure
SENSOR_FAILURE	Bad_SensorFailure
LAST_KNOWN	Bad_OutOfService
OUT_OF_SERVICE	Bad_OutOfService
WAITING_FOR_INITIAL_DATA	Bad_WaitingForInitialData

A.3.2.4 Timestamp

The Timestamp provided for a value in the DA server is assigned to the SourceTimeStamp of the DataValue in the COM UA Wrapper.

The ServerTimeStamp in the DataValue is set to the current time by the COM UA Wrapper at the start of the Read Operation.

A.3.3 Read data

The COM UA Wrapper supports performing Read operations to DA servers of versions 2.05a and 3.

For version 2.05a, the COM UA wrapper creates a Group using the IOPCServer::AddGroup method and adds the items whose data is to be read to the Group using IOPCItemMgmt::AddItems method. The Data is retrieved for the items using the IOPCSyncIO::Read method. The VQT for each item is mapped to the DataValue structure as shown in Figure A.2. Please note that only Read from Device is supported for this version. The "maxAge" parameter is ignored.

For version 3, the COM UA Wrapper uses the IOPCItemIO::Read to retrieve the data. The VQT for each item is mapped to the DataValue structure as shown in Figure A.2. The Read supports both the Read from Device and Cache and uses the "maxAge" parameter.

If there are errors for the items in the Read from the DA server, then these are mapped to the StatusCode of the DataValue in the COM UA Wrapper.

The mapping of the OPC COM DA Read Errors code to OPC UA Status code (in the COM UA Wrapper) is shown in Table A.4:

Table A.4 – OPC DA Read error mapping

OPC DA Error ID	OPC UA Status Code
OPC_E_BADRIGHTS	Bad_NotReadable
E_OUTOFMEMORY	Bad_OutOfMemory
OPC_E_INVALIDHANDLE	Bad_NodeIdUnknown
OPC_E_UNKNOWNITEMID	Bad_NodeIdUnknown
E_INVALIDITEMID	Bad_NodeIdInvalid
E_INVALID_PID	Bad_AttributeIdInvalid
E_ACCESSDENIED	Bad_OutOfService
Others	Bad_UnexpectedError

A.3.4 Write Data

The COM UA Wrapper supports performing Write operations to DA servers of versions 2.05a and 3.

For version 2.05a, the COM UA wrapper creates a Group using the IOPCServer::AddGroup method and adds the items whose data is to be written using IOPCItemMgmt::AddItems method. The value is written for the items using the IOPCSyncIO::Write method. If the StatusCode or TimeStamps (Source or Server) is specified to be written for the item, then the COM UA Wrapper returns a BadWriteNotSupported Status code for the item.

For version 3, the COM UA Wrapper uses the IOPCItemIO::WriteVQT data including StatusCode and TimeStamp.If a SourceTimeStamp is provided, this timestamp is used for the Write else the ServerTimeStamp is used.

If there are errors for the items in the Write from the DA server, then these are mapped to the StatusCode for the corresponding item.

The mapping of the OPC COM DA Write Errors code to OPC UA Status code (in the COM UA Wrapper) is shown in Table A.5.

Table A.5 – OPC DA Write error code mapping

OPC DA Error ID	OPC UA Status Code
E_BADRIGHTS	Bad_NotWritable
DISP_E_TYPEMISMATCH	Bad_TypeMismatch
E_BADTYPE	Bad_TypeMismatch
E_RANGE	Bad_OutOfRange
DISP_E_OVERFLOW	Bad_OutOfRange
E_OUTOFMEMORY	Bad_OutOfMemory
E_INVALIDHANDLE	Bad_NodeIdUnknown
E_UNKNOWNITEMID	Bad_NodeIdUnknown
E_INVALIDITEMID	Bad_NodeIdInvalid
E_INVALID_PID	Bad_NodeIdInvalid
E_NOTSUPPORTED	Bad_WriteNotSupported
S_CLAMP	Good_Clamped
Others	Bad_UnexpectedError

A.3.5 Subscriptions

A subscription is created in the DA server when a MonitoredItem is created in the COM UA Wrapper.

The SamplingInterval and the Deadband value are used for the subscription to setup a periodic data change call back on the COM UA Wrapper. Note that only the PercentDeadbandType is supported by the COM UA Wrapper.

The VQT for each item is mapped to the DataValue structure as shown in Figure A.2 and published to the client by the COM UA Wrapper periodically.

The mapping of the OPC COM DA Read Errors code to OPC UA Status code (in the COM UA Wrapper) is the same as the Read mapping in Figure A.2.

A.4 COM UA proxy for DA Client

A.4.1 Guidelines

The Data Access COM UA Proxy is a COM Server combined with a UA Client. It maps the Data Access address space of UA Data Access Server into the appropriate COM Data Access objects.

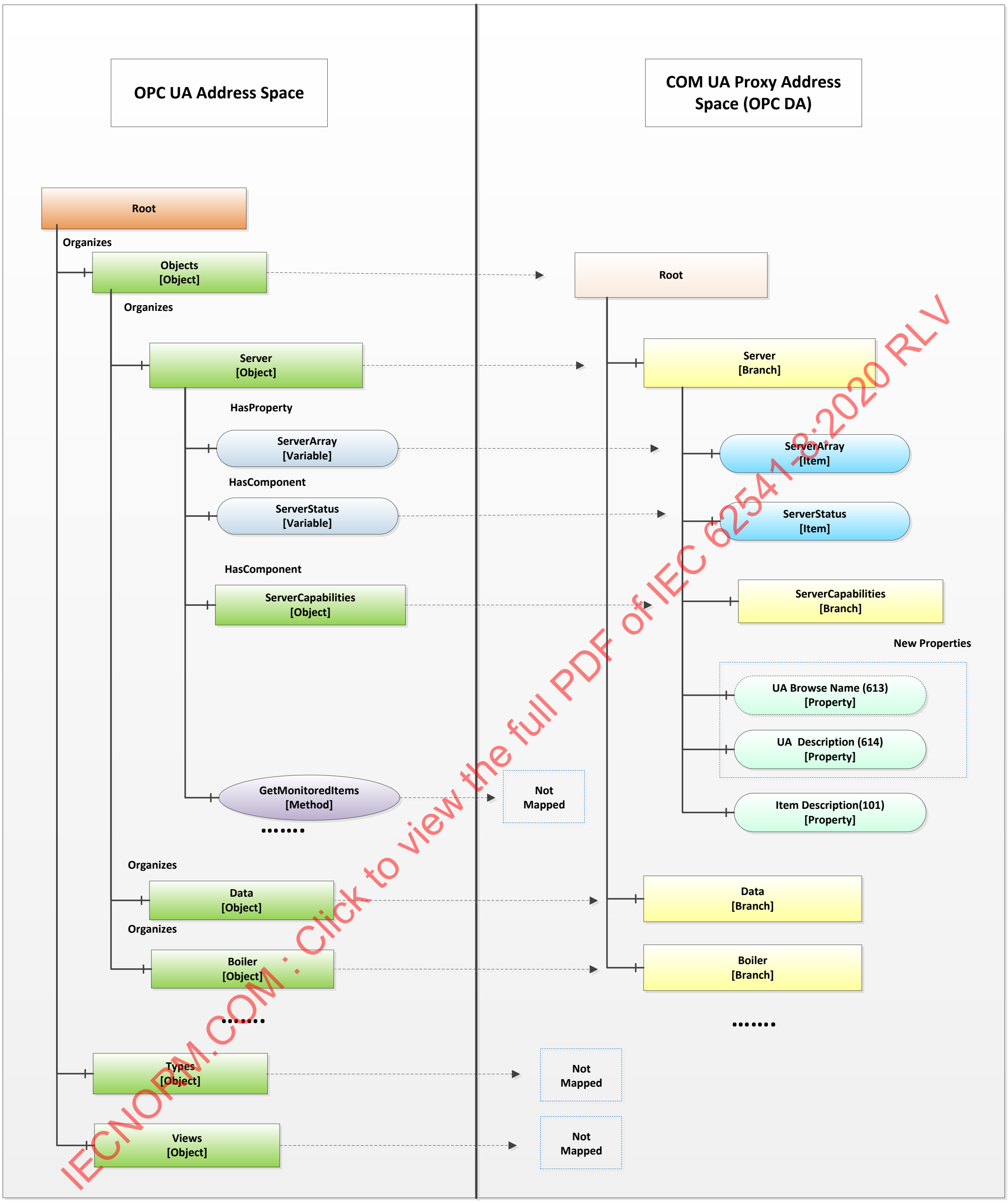
Sublauses A.4.1 through A.4.6 identify the design guidelines and constraints used to develop the Data Access COM UA Proxy provided by the OPC Foundation. In order to maintain a high degree of consistency and interoperability, it is strongly recommended that vendors, who choose to implement their own version of the Data Access COM UA Proxy, follow these same guidelines and constraints.

The Data Access COM Client simply needs to address how to connect to the UA Data Access Server. Connectivity approaches include the one where Data Access COM Clients connect to a UA Data Access Server with a CLSID just as if the target Server were a Data Access COM Server. However, the CLSID can be considered virtual since it is defined to connect to intermediary components that ultimately connect to the UA Data Access Server. Using this approach, the Data Access COM Client calls co-create instance with a virtual CLSID as described above. This connects to the Data Access COM UA Proxy components. The Data Access COM UA Proxy then establishes a secure channel and session with the UA Data Access Server. As a result, the Data Access COM Client gets a COM Data Access Server interface pointer.

A.4.2 Information Model and Address Space mapping

A.4.2.1 General

OPC UA defines 8 Node Class types in the address space Object, Variable, Method, ObjectType, VariableType, ReferenceType, DataType, View. The COM UA Proxy maps only the nodes of Node Class types Object, Variable to the OPC DA types as shown in the figure below. Only the nodes under the Objects node are considered for the COM UA Proxy address space and others such as Types and Views are not mapped. Figure A.4 shows an example mapping of OPC DA to OPC UA information.



IEC

Figure A.4 – Sample OPC DA mapping of OPC UA Information Model and Address Space

A.4.2.2 Object Nodes

A node of Object Node class in the OPC UA server is represented in the Data Access COM UA Proxy as a Branch.

The root of the Data Access COM UA Proxy is the Objects folder of the OPC UA Server.

The OPC UA Address space hierarchy is discovered using the Browse Service for the Objects Node using the following filters:

- BrowseDirection as Forward;
- ReferenceTypeId as Organizes and HasChild;
- IncludeSubtypes as True;
- NodeClassMask as Object and Variable.

The DisplayName of the OPC UA node is used as the Name for each Branch in the Data Access COM UA Proxy

Each Branch in the Data Access COM UA Proxy is assigned 3 properties:

- *UA Browse Name* (Property ID: 613): the value of the *BrowseName* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Description* (Property ID: 614): the value of the *Description* attribute of the node in the OPC UA Server is assigned to this property, if a Description attribute is provided.
- *Item Description* (Property ID: 101): the value of the *DisplayName* attribute of the node in the OPC UA Server is assigned to this property.

NOTE COM DA Clients typically display the ItemID and the Item Description. Since the ItemID generated by the UA Proxy may be particularly difficult to read and understand, proxies may use the DisplayName as value for the Item Description Property as it will be easier to understand by a human user.

A.4.2.3 Variable Nodes

A node of Variable Node class in the OPC UA server is represented in the Data Access COM UA Proxy as an Item.

The DisplayName of the OPC UA node is used as the Name for each Item in the Data Access COM UA Proxy.

The NodeId of the OPC UA node is used as the ItemId for each Item in the Data Access COM UA Proxy, but the ‘=" character is replaced with ‘-’ in the string. For example, NodeId: ns=4,i=10, ItemID = "ns-4;i-10" or NodeId: ns=4,s=FL102, ItemID = "ns-4,s-FL102"

Each Item in the Data Access COM UA Proxy is assigned the following properties based on the node attributes or its references:

Standard Properties:

- *Item Canonical Data Type* (Property ID: 1): the combined value of the *DataType* attribute and the *ValueRank* attribute of the node in the OPC UA Server is assigned to this property (see A.4.3.2).
- *Item Value* (Property ID: 2): the value of the *Value* attribute of the node in the OPC UA Server is assigned to this property. Details on Value mapping are in A.4.3.2.
- *Item Quality* (Property ID: 3): the *StatusCode* of the *Value* obtained for the node in the OPC UA Server is assigned to this property. Details on Quality mapping are in A.4.3.3.
- *Item Timestamp* (Property ID: 4): the *SourceTimestamp or ServerTimestamp* of the *Value* obtained for the node in the OPC UA Server is assigned to this property. Details on Timestamp mapping are in A.4.3.4.
- *Item Access Rights* (Property ID: 5): the value of the *AccessLevel* attribute of the node in the OPC UA Server is assigned to this property based on the following mapping:
  - CurrentRead -> OPC_READABLE
  - CurrentWrite -> OPC_WRITABLE

  The other AccessLevel provided by OPC are ignored
- *Server Scan Rate* (Property ID: 6): the value of the *MinimumSamplingInterval* attribute of the node in the OPC UA Server is assigned to this property.

- *Item EU Type* (Property ID: 7): the EU Type value is assigned based on the references of the node in the OPC UA Server:
  - *Analog(1)*: if the node in the OPC UA Server references a *EURange property* node, then it is assigned the *Analog EU Type.*
  - *Enumerated(2)*: if the node in the OPC UA Server references a *EnumStrings property* node, then it is assigned the *Enumerated EU Type.*
  - *Empty(0)*: For a node in the OPC UA Server that does not meet above criteria, the type is set as 0 (Empty)
- *EU Info* (Property ID: 8): if the node in the OPC UA Server references an *EnumStrings property* node, then the enumerated values of the property node is assigned to this property.
- *EU Units* (Property ID: 100): if the node in the OPC UA Server references a *EngineeringUnits property* node, then the value of the *EngineeringUnits* property node is assigned the *EU Units* property*.*
- *Item Description* (Property ID: 101): The value of the *DisplayName* attribute of the node in the OPC UA Server is assigned to this property.
- *High EU* (Property ID: 102): if the node in the OPC UA Server references a *EURange property* node, then the *'High'* value of the property node is assigned to this property*.*
- *Low EU* (Property ID: 103): if the node in the OPC UA Server references a *EURange property* node, then the *'Low'* value of the property node is assigned to this property.
- *High Instrument Range* (Property ID: 104): if the node in the OPC UA Server references an *InstrumentRange property* node, then the *'High'* value of the property node is assigned to this property*.*
- *Low Instrument Range* (Property ID: 105): if the node in the OPC UA Server references an *InstrumentRange property* node, then the *'Low'* value of the property node is assigned to this property*.*
- *Contact Close Label* (Property ID: 106): if the node in the OPC UA Server references a *FalseState property* node, then the value of the property node is assigned to this property*.*
- *Contact Open Label* (Property ID: 107): if the node in the OPC UA Server references a *TrueState property* node, then the value of the property node is assigned to this property*.*
- *Item Time Zone* (Property ID: 108): if the node in the OPC UA Server references a *TimeZone property* node, then the 'Offset' value of the property node is assigned to this property*.*

New Properties:

- *UA BuiltIn Type* (Property ID: 610): the identifier value of the *DataType* node associated with the DataType attribute of the node in the OPC UA Server is assigned to this property.
- *UA Data Type Id* (Property ID: 611): the complete NodeId value (namespace and identifier) of the *DataType* node associated with the DataType attribute of the node in the OPC UA Server is assigned to this property.
- *UA Value Rank* (Property ID: 612): the value of the *ValueRank* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Browse Name* (Property ID: 613): the value of the *BrowseName* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Description* (Property ID: 614): the value of the *Description* attribute of the node in the OPC UA Server is assigned to this property.

A.4.2.4 Namespace Indices

For generating ItemIDs, the Proxy uses Namespace Indices. To assure that Clients can persist these ItemIDs, the Namespace Indices shall never change. To accomplish this the Proxy has to persist its Namespace Table and only append entries but never change existing ones.

The Proxy shall also provide a translation from the current Namespace Table in the Server to the persisted Namespace Table.

If you move or copy the Proxy to another machine, the Namespace Table has to be copied to this machine as well.

A.4.3 Data and error mapping

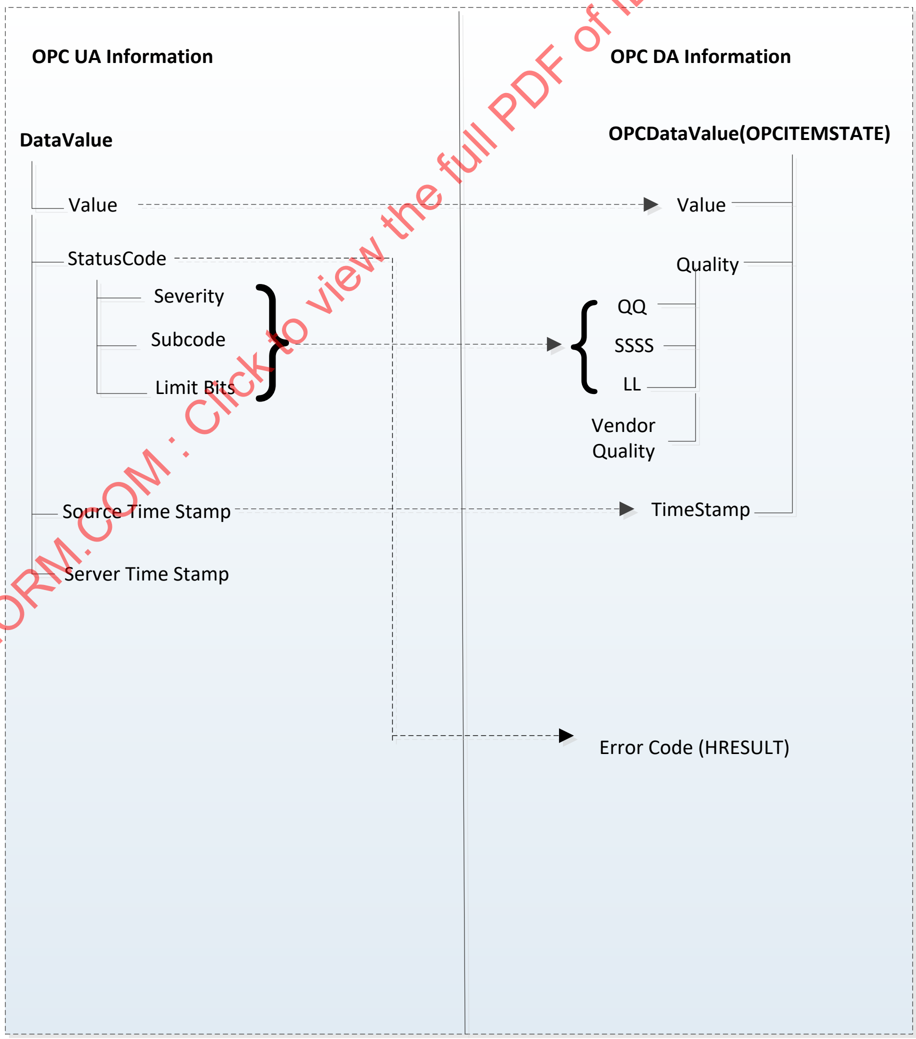
A.4.3.1 General

In an OPC UA Server, Automation Data is represented as a Data Value and and status, in addition additional error data can be provided via Diagnostic Info for a tag

The COM UA Proxy maps the Data Value structure into VQT data and error code.

For successful operations (StatusCode of Good and Uncertain), the COM UA Proxy maps the Status Code of the DataValue to the OPC DA Quality But in case of error (StatusCode of Bad), the Status Code is mapped to the OPC DA Error code.

The StatusCode in the Diagnostic Info returned by the OPC UA Server are mapped to OPC DA Error codes. Figure A.5 illustrates this mapping.



IEC

Figure A.5 – OPC UA to OPC DA data & error mapping

A.4.3.2 Value

The COM UA Proxy converts the OPC UA Data Value to the corresponding OPC DA Variant type. The mapping is shown in Table A.6. For DataTypes that are subtypes of an existing base DataType the conversion for the Base DataType is used.

Table A.6 – DataTypes and Mapping

OPC UA Data type (Bin UA Server)	Variant Data Type (In DA server)
Int16	VT_I2
Int32	VT_I4
Float	VT_R4
Double	VT_R8
Decimal	VT_DECIMAL
String	VT_BSTR
Boolean	VT_BOOL
Byte	VT_UI1
SByte	VT_I1
UInt16	VT_UI2
UInt32	VT_UI4
Int64	VT_I8
UInt64	VT_UI8
Guid	VT_BSTR
DateTime	VT_DATE
NodeId	VT_BSTR
XmlElement	VT_BSTR
ExpandedNodeId	VT_BSTR
QualifiedName	VT_BSTR
LocalizedText	VT_BSTR
StatusCode	VT_UI4
ExtensionObject	Array of VT_UI1
Array of above OPC UA types	Array of corresponding Variant type

A.4.3.3 Quality

The Quality of a Data Value in the OPC UA Server is represented as a StatusCode.

The COM UA Proxy maps the Severity, Subcode and the limit bits of the OPC UA Status code to the lower 8 bits of the OPC DA Quality structure (of the form QQSSSSLL). Figure A.6 illustrates this mapping.

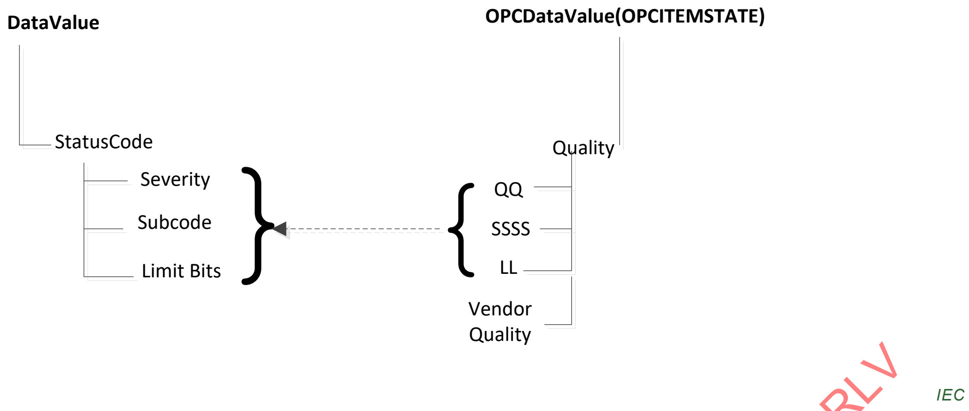


Figure A.6 – OPC UA Status Code to OPC DA quality mapping

The Severity field of the Status code is mapped to the primary quality. The SubCode is mapped to the Sub Status and the Limit Bits are mapped to the Limit field.

Table A.7 shows a mapping of the OPC UA status code to OPC DA primary quality.

Table A.7 – Quality mapping

OPC UA Status Code	OPC DA Primary Quality (Quality & Sub status QQSSSS)
Good	GOOD
Good_LocalOverride	LOCAL_OVERRIDE
Uncertain	UNCERTAIN
Uncertain_SubNormal	SUB_NORMAL
Uncertain_SensorNotAccurate	SENSOR_CAL
Uncertain_EngineeringUnitsExceeded	EGU_EXCEEDED
Uncertain_LastUsableValue	LAST_USABLE
Bad	BAD
Bad_ConfigurationError	CONFIG_ERROR
Bad_NotConnected	NOT_CONNECTED
Bad_NoCommunication	COMM_FAILURE
Bad_OutOfService	OUT_OF_SERVICE
Bad_DeviceFailure	DEVICE_FAILURE
Bad_SensorFailure	SENSOR_FAILURE
Bad_WaitingForInitialData	WAITING_FOR_INITIAL_DATA

A.4.3.4 Timestamp

If available, the SourceTimestamp of the DataValue in the OPC UA Server is assigned to the Timestamp for the value in the COM UA Proxy. If SourceTimestamp is not available, then the ServerTimestamp is used.

A.4.4 Read data

The COM UA Proxy converts all the ItemIds in the Read into valid NodeIds by replacing the ‘-’ with ‘=’ and calls the OPC UA Read Service for the Value Attribute.

If the Read Service call is successful, then DataValue for each node is mapped to the VQT for each item as shown in Figure A.5.

If the Read Service call fails or if there are errors for some of the Nodes, then the StatusCodes of these Nodes are mapped to the error code by the COM UA Proxy.

The mapping of the OPC UA Status code to OPC DA Read Error code (in the COM UA Proxy) is shown in Table A.8:

Table A.8 – OPC UA Read error mapping

OPC UA Status Code	OPC DA Error ID
Bad_OutOfMemory	E_OUTOFMEMORY
Bad_NodeIdInvalid	E_INVALIDITEMID
Bad_NodeIdUnknown	E_UNKNOWNITEMID
Bad_NotReadable	E_BADRIGHTS
Bad_UserAccessDenied	E_ACCESSDENIED
Bad_AttributeIdInvalid	E_INVALIDITEMID
Bad_UnexpectedError	E_FAIL
Bad_InternalError	E_FAIL
Bad_SessionClosed	E_FAIL
Bad_TypeMismatch	E_BADTYPE

A.4.5 Write data

The COM UA Proxy converts all the ItemIds in the Write into valid NodeIds by replacing the '-' with '='. It converts the Value, Quality and Timestamp (VQT) to a DataValue structure as per the mapping in Figure A.5, and calls the OPC UA Write Service for the Value Attribute.

If the Write Service call fails or if there are errors for some of the Nodes, then the StatusCodes of these Nodes are mapped to the error code by the COM UA Proxy.

The mapping of the OPC UA Status code to OPC DA Write Error code (in the COM UA Proxy) is shown in Table A.9.

Table A.9 – OPC UA Write error code mapping

OPC UA Status Code	OPC DA Error ID
Bad_TypeMismatch	E_BADTYPE
Bad_OutOfMemory	E_OUTOFMEMORY
Bad_NodeIdInvalid	E_INVALIDITEMID
Bad_NodeIdUnknown	E_UNKNOWNITEMID
Bad_NotWritable	E_BADRIGHTS
Bad_UserAccessDenied	E_ACCESSDENIED
Bad_AttributeIdInvalid	E_UNKNOWNITEMID
Bad_WriteNotSupported	E_NOTSUPPORTED
Bad_OutOfRange	E_RANGE

A.4.6 Subscriptions

The COM UA Proxy creates a Subscription in the OPC UA Server when a Group is created. The Name, Active flag, UpdateRate parameters of the Group are used while creating the subscription.

The COM UA Proxy Creates Monitored Items in the OPC UA Server when items are added to the Group.

The following parameters and filters are used for creating the monitored items:

- The *ItemIds* are converted to valid NodeIds by replacing the ‘-’ with ‘=’.
- Data Change Filter is used for Items with EU type as "Analog":
  - Trigger = STATUS_VALUE_1
  - if DeadBand value is specified for the *Group*:
    - DeadbandType = Percent_2
    - DeadbandValue = deadband specified for the group.

The COM UA Proxy calls the Publish Service of the OPC UA Server periodically and sends any data changes to the client.

IEC 62541-8

Edition 3.0 2020-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 8: Data access

Architecture unifiée OPC – Partie 8: Accès aux données

IEC 62541-8:2020-06(en-fr)

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

OPC UNIFIED ARCHITECTURE –

Part 8: Data access

FOREWORD

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

International Standard IEC 62541-8 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

This third edition cancels and replaces the second edition published in 2015. This edition constitutes a technical revision.

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

a) added new VariableTypes for AnalogItems;
b) added an Annex that specifies a recommended mapping of OPC UA Dataccess to OPC COM DataAccess;
c) changed the ambiguous description of "Bad_NotConnected";
d) updated description for EUInformation to refer to latest revision of UNCEFACT units.

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

FDIS	Report on voting
65E/708/FDIS	65E/726/RVD

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

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

Throughout this document and the other parts of the IEC 62541 series, certain document conventions are used:

Italics are used to denote a defined term or definition that appears in the "Terms and definition" clause in one of the parts of the IEC 62541 series.

Italics are also used to denote the name of a service input or output parameter or the name of a structure or element of a structure that are usually defined in tables.

The *italicized terms and names* are, with a few exceptions, written in camel-case (the practice of writing compound words or phrases in which the elements are joined without spaces, with each element's initial letter capitalized within the compound). For example, the defined term is *AddressSpace* instead of Address Space. This makes it easier to understand that there is a single definition for *AddressSpace*, not separate definitions for Address and Space.

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, 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 "http://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 publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

OPC UNIFIED ARCHITECTURE –

Part 8: Data access

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture (OPC UA) standard series and defines the information model associated with Data Access (DA). It particularly includes additional *VariableTypes* and complementary descriptions of the *NodeClasses* and *Attributes* needed for Data Access, additional *Properties,* and other information and behaviour.

The complete address space model, including all *NodeClasses* and *Attributes* is specified in IEC 62541-3. The services to detect and access data are specified in IEC 62541-4.

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 TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

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*

UN/CEFACT: UNECE Recommendation N° 20, *Codes for Units of Measure Used in International Trade*, available at
https://www.unece.org/cefact/codesfortrade/codes_index.html

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC 62541-3, and IEC 62541-4 and the following apply.

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

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

3.1.1
DataItem
link to arbitrary, live automation data, that is, data that represents currently valid information

Note 1 to entry: Examples of such data are

- device data (such as temperature sensors),

- calculated data,
- status information (open/closed, moving),
- dynamically changing system data (such as stock quotes),
- diagnostic data.

3.1.2
AnalogItem

DataItem that represents continuously variable physical quantities (e.g. length, temperature), in contrast to the digital representation of data in discrete items

Note 1 to entry: Typical examples are the values provided by temperature sensors or pressure sensors. OPC UA defines a specific *VariableType* to identify an *AnalogItem*. *Properties* describe the possible ranges of *AnalogItems*.

3.1.3
DiscreteItem

DataItem that represents data that may take on only a certain number of possible values (e.g. OPENING, OPEN, CLOSING, CLOSED)

Note 1 to entry: Specific *VariableTypes* are used to identify *DiscreteItems* with two states or with multiple states. *Properties* specify the string values for these states.

3.1.4
ArrayItem

DataItem that represents continuously variable physical quantities and where each individual data point consists of multiple values represented by an array (e.g., the spectral response of a digital filter)

Note 1 to entry: Typical examples are the data provided by analyser devices. Specific *VariableTypes* are used to identify *ArrayItem* variants.

3.1.5
EngineeringUnits

units of measurement for *AnalogItems* that represent continuously variable physical quantities (e.g. length, mass, time, temperature)

Note 1 to entry: This standard defines *Properties* to inform about the unit used for the *DataItem* value and about the highest and lowest value likely to be obtained in normal operation.

3.2 Abbreviated terms

DA	data access
EU	engineering unit
UA	Unified Architecture

4 Concepts

Data Access deals with the representation and use of automation data in Servers.

Automation data can be located inside the *Server* or on I/O cards directly connected to the *Server*. It can also be located in sub-servers or on other devices such as controllers and input/output modules, connected by serial links via field buses or other communication links. OPC UA Data Access *Servers* provide one or more OPC UA Data Access *Clients* with transparent access to their automation data.

The links to automation data instances are called *DataItems*. The categories of automation data are provided is completely vendor-specific. Figure 1 illustrates how the *AddressSpace* of a *Server* may consist of a broad range of different *DataItems*.

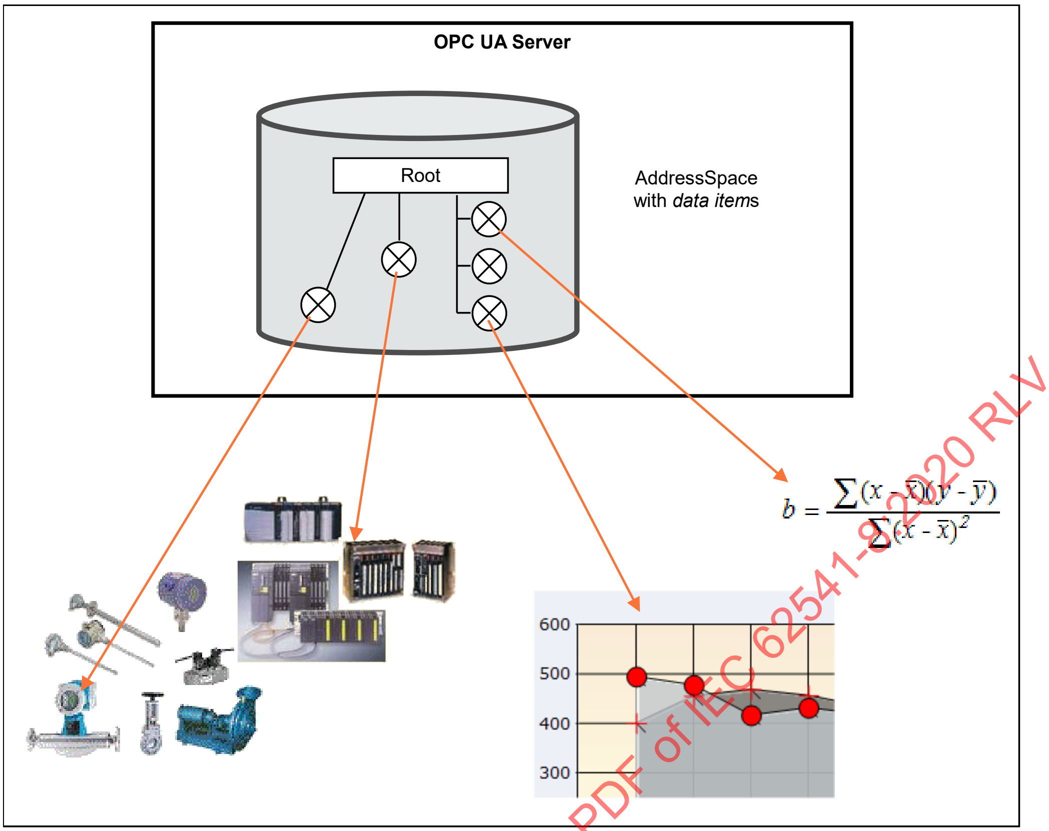


Figure 1 – OPC *DataItems* are linked to automation data

Clients may read or write *DataItems*, or monitor them for value changes. The *Services* needed for these operations are specified in IEC 62541-4. Changes are defined as a change in status (quality) or a change in value that exceeds a client-defined range called a *Deadband*. To detect the value change, the difference between the current value and the last reported value is compared to the *Deadband*.

5 Model

5.1 General

The DataAccess model extends the variable model by defining *VariableTypes*. The *DataItemType* is the base type. *ArrayItemType*, *BaseAnalogType* and *DiscreteItemType* are specializations. See Figure 2. Each of these *VariableTypes* can be further extended to form domain- or server-specific *DataItems*.

Annex A specifies the recommended way for mapping the information received from OPC COM Data Access (DA) Servers to the model in this document.

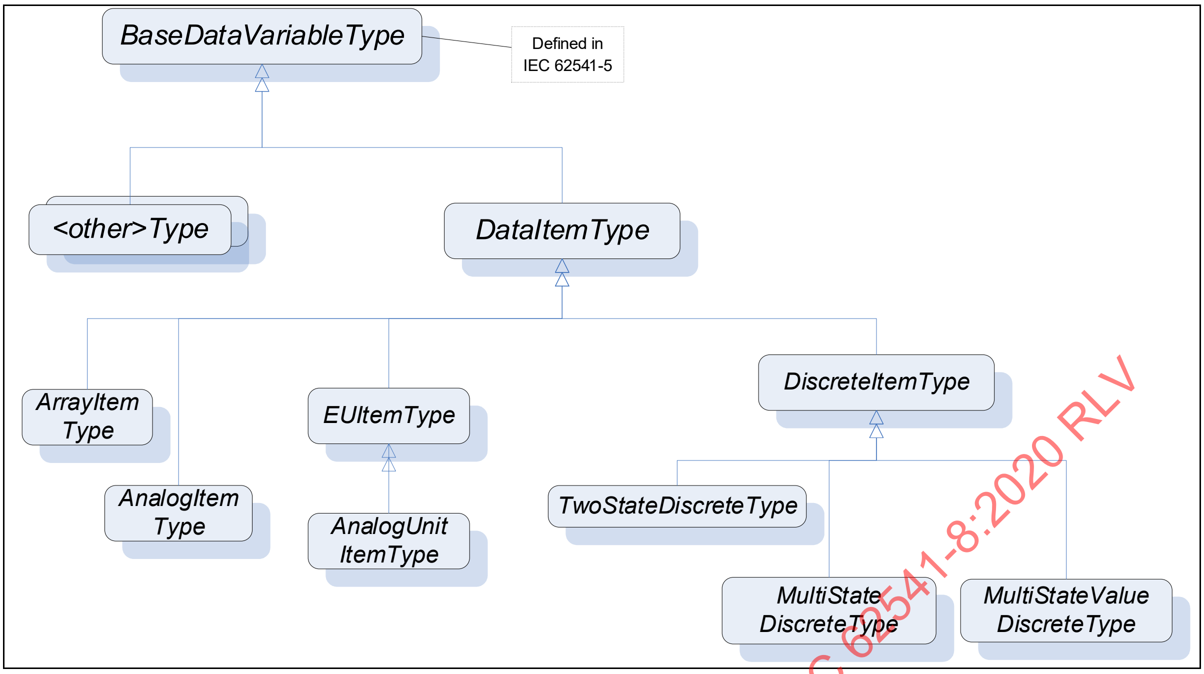


Figure 2 – DataItem *VariableType* hierarchy

5.2 SemanticsChanged

The *StatusCode* also contains an informational bit called *SemanticsChanged*.

Servers that implement Data Access shall set this Bit in notifications if certain *Properties* defined in this standard change. The corresponding *Properties* are specified individually for each *VariableType*.

Clients that use any of these *Properties* should re-read them before they process the data value.

5.3 Variable Types

5.3.1 DataItemType

This *VariableType* defines the general characteristics of a *DataItem*. All other *DataItem* Types derive from it. The *DataItemType* derives from the *BaseDataVariableType* and therefore shares the variable model as described in IEC 62541-3 and IEC 62541-5. It is formally defined in Table 1.

Table 1 – DataItemType definition

Attribute	Value				
BrowseName	DataItemType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseDataVariableType* defined in IEC 62541-5; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogItemType	Defined in 5.3.2		
HasSubtype	VariableType	DiscreteItemType	Defined in 5.3.3		
HasSubtype	VariableType	ArrayItemType	Defined in 5.3.4		
HasProperty	Variable	Definition	String	PropertyType	Optional
HasProperty	Variable	ValuePrecision	Double	PropertyType	Optional

Definition is a vendor-specific, human-readable string that specifies how the value of this *DataItem* is calculated. *Definition* is non-localized and will often contain an equation that can be parsed by certain clients.

EXAMPLE: *Definition*::= "(TempA – 25) + TempB"

ValuePrecision specifies the maximum precision that the *Server* can maintain for the item based on restrictions in the target environment.

ValuePrecision can be used for the following *DataTypes*:

- for Float and Double values it specifies the number of digits after the decimal place;
- for DateTime values it indicates the minimum time difference in nanoseconds. For example, a ValuePrecision of 20 000 000 defines a precision of 20 ms.

The *ValuePrecision Property* is an approximation that is intended to provide guidance to a *Client*. A *Server* is expected to silently round any value with more precision that it supports. This implies that a *Client* may encounter cases where the value read back from a *Server* differs from the value that it wrote to the *Server*. This difference shall be no more than the difference suggested by this *Property*.

5.3.2 AnalogItem VariableTypes

5.3.2.1 General

The *VariableTypes* in this subclause define the characteristics of *AnalogItems*. The types have identical semantics and *Properties* but with diverging *ModellingRules* for individual *Properties*.

The *Properties* are only described once – in 5.3.2.2. The descriptions apply to the *Properties* for the other *VariableTypes* as well.

5.3.2.2 BaseAnalogType

This *VariableType* is the base type for analog items. All *Properties* are optional. Subtypes of this base type will mandate some of the *Properties*. The *BaseAnalogType* derives from the *DataItemType*. It is formally defined in Table 2.

Table 2 – BaseAnalogType definition

Attribute	**Value**				
BrowseName	BaseAnalogType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.3.1 i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogItemType	Defined in 5.3.2.3		
HasSubtype	VariableType	AnalogUnitType	Defined in 5.3.2.4		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Optional
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional

The following paragraphs describe the *Properties* of this *VariableType*. If the analog item's *Value* contains an array, the *Properties* shall apply to all elements in the array.

InstrumentRange defines the value range that can be returned by the instrument.

EXAMPLE 1 *InstrumentRange*::= {−9 999,9, 9 999,9}

Although defined as optional, it is strongly recommended for *Servers* to support this *Property*. Without an *InstrumentRange* being provided, *Clients* will commonly assume the full range according to the *DataType*.

The *InstrumentRange Property* may also be used to restrict a Built-in DataType such as Byte or Int16) to a smaller range of values.

EXAMPLE 2

Uint4:	*InstrumentRange*::= {0, 15}
Int6:	*InstrumentRange*::= {−32, 31}

The *Range Data Type* is specified in 5.6.2.

EURange defines the value range likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.

Sensor or instrument failure or deactivation can result in a returned item value which is actually outside of this range. *Client* software shall be prepared to deal with this possibility. Similarly a *Client* may attempt to write a value that is outside of this range back to the server. The exact behaviour (accept, reject, clamp, etc.) in this case is *Server*-dependent. However, in general *Servers* shall be prepared to handle this.

EXAMPLE 3 *EURange*::= {−200,0, 1 400,0}

See also 6.2 for a special monitoring filter (*PercentDeadband*) which is based on the engineering unit range.

NOTE 1 If *EURange* is not provided on an instance, the *PercentDeadband* filter cannot be used for that instance (see 6.2).

EngineeringUnits specifies the units for the *DataItem*'s value (e.g., DEGC, hertz, seconds). The *EUInformation* type is specified in 5.6.3.

It is important to note that understanding the units of a measurement value is essential for a uniform system. In an open system in particular where *Servers* from different cultures might be used, it is essential to know what the units of measurement are. Based on such knowledge, values can be converted if necessary before being used. Therefore, although defined as optional, support of the *EngineeringUnits Property* is strongly advised.

OPC UA recommends using the "Codes for Units of Measurement" (see UN/CEFACT: UNECE Recommendation N° 20). The mapping to the *EngineeringUnits Property* is specified in 5.6.3.

NOTE 2 Examples for unit mixup: in 1999, the Mars Climate Orbiter crashed into the surface of Mars. The main reason was a discrepancy over the units used. The navigation software expected data in newton second; the company who built the orbiter provided data in pound-force seconds. Another, less expensive, disappointment occurs when people used to British pints order a pint in the USA, only to be served what they consider a short measure.

The *StatusCode SemanticsChanged* bit shall be set if any of the *EURange* (could change the behaviour of a *Subscription* if a *PercentDeadband* filter is used) or *EngineeringUnits* (could create problems if the *Client* uses the value to perform calculations) *Properties* are changed (see 5.2 for additional information).

5.3.2.3 AnalogItemType

This *VariableType* requires the *EURange Property*. The *AnalogItemType* derives from the *BaseAnalogType*. It is formally defined in Table 3.

Table 3 – AnalogItemType definition

Attribute	**Value**				
BrowseName	AnalogItemType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseAnalogType* defined in 5.3.2.2, i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	AnalogUnitRangeType	Defined in 5.3.2.5		
HasProperty	Variable	EURange	Range	PropertyType	Mandatory

5.3.2.4 AnalogUnitType

This *VariableType* requires the *EngineeringUnits Property*. The *AnalogUnitType* derives from the *BaseAnalogType*. It is formally defined in Table 4.

Table 4 – AnalogUnitType definition

Attribute	**Value**				
BrowseName	AnalogUnitType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseAnalogType* defined in 5.3.2.2, i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.2.5 AnalogUnitRangeType

The *AnalogUnitRangeType* derives from the *AnalogItemType* and additionaly requires the *EngineeringUnits Property*. It is formally defined in Table 5.

Table 5 – AnalogUnitRangeType definition

Attribute	**Value**				
BrowseName	AnalogUnitRangeType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AnalogItemType* defined in 5.3.2.3, i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.3 DiscreteItemType

5.3.3.1 General

This VariableType is an abstract type. That is, no instances of this type can exist. However, it might be used in a filter when browsing or querying. The *DiscreteItemType* derives from the *DataItemType* and therefore shares all of its characteristics. It is formally defined in Table 6.

Table 6 – DiscreteItemType definition

Attribute	**Value**				
BrowseName	DiscreteItemType				
IsAbstract	True				
ValueRank	−2 (−2 = 'Any')				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.2; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	TwoStateDiscreteType	Defined in 5.3.3.2		
HasSubtype	VariableType	MultiStateDiscreteType	Defined in 5.3.3.3		
HasSubtype	VariableType	MultiStateValueDiscreteType	Defined in 5.3.3.4		

5.3.3.2 TwoStateDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have two states. The *TwoStateDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 7.

Table 7 – TwoStateDiscreteType definition

Attribute	Value				
BrowseName	TwoStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	Boolean				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	TrueState	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	FalseState	LocalizedText	PropertyType	Mandatory

TrueState contains a string to be associated with this *DataItem* when it is TRUE. This is typically used for a contact when it is in the closed (non-zero) state.

for example: "RUN", "CLOSE", "ENABLE", "SAFE", etc.

FalseState contains a string to be associated with this *DataItem* when it is FALSE. This is typically used for a contact when it is in the open (zero) state.

for example: "STOP", "OPEN", "DISABLE", "UNSAFE", etc.

If the item contains an array, then the *Properties* will apply to all elements in the array.

The *StatusCode SemanticsChanged* bit shall be set if any of the *FalseState or TrueState (*changes can cause misinterpretation by users or (scripting) programs*) Properties* are changed (see 5.2 for additional information).

5.3.3.3 MultiStateDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have more than two states. The *MultiStateDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 8.

Table 8 – MultiStateDiscreteType definition

Attribute	Value				
BrowseName	MultiStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = ‘Any’)				
DataType	UInteger				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EnumStrings	LocalizedText[]	PropertyType	Mandatory

EnumStrings is a string lookup table corresponding to sequential numeric values (0, 1, 2, etc.)

Example:

"OPEN"
"CLOSE"
"IN TRANSIT" etc.

Here the string "OPEN" corresponds to 0, "CLOSE" to 1 and "IN TRANSIT" to 2.

Clients should be prepared to handle item values outside of the range of the list; and robust servers should be prepared to handle writes of illegal values.

If the item contains an array, then this lookup table shall apply to all elements in the array.

NOTE The *EnumStrings* property is also used for Enumeration *DataTypes* (for the specification of this *DataType*, see IEC 62541-3).

The *StatusCode SemanticsChanged* bit shall be set if the *EnumStrings (*changes can cause misinterpretation by users or (scripting) programs*) Property* is changed (see 5.2 for additional information).

5.3.3.4 MultiStateValueDiscreteType

This *VariableType* defines the general characteristics of a *DiscreteItem* that can have more than two states and where the state values (the enumeration) does not consist of consecutive numeric values (may have gaps) or where the enumeration is not zero-based. The *MultiStateValueDiscreteType* derives from the *DiscreteItemType*. It is formally defined in Table 9.

Table 9 – MultiStateValueDiscreteType definition

Attribute	**Value**				
BrowseName	MultiStateValueDiscreteType				
IsAbstract	False				
ValueRank	Scalar				
DataType	Number				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DiscreteItemType* defined in 5.3.3; i.e the *Properties* of that type are inherited.					
HasProperty	Variable	EnumValues	See IEC 62541-3		Mandatory
HasProperty	Variable	ValueAsText	See IEC 62541-3		Mandatory

EnumValues is an array of *EnumValueType*. Each entry of the array represents one enumeration value with its integer notation, a human-readable representation, and help information. This represents enumerations with integers that are not zero-based or have gaps (e.g. 1, 2, 4, 8, 16). See IEC 62541-3 for the definition of this type. *MultiStateValueDiscrete Variables* expose the current integer notation in their *Value Attribute*. *Clients* will often read the *EnumValues Property* in advance and cache it to lookup a name or help whenever they receive the numeric representation.

Only *DataTypes* that can be represented with *EnumValues* are allowed for *Variables* of *MultiStateValueDiscreteType*. These are*:*

- signed integers up to 64 bits in length;
- unsigned integers up to 63 bits in length.

The numeric representation of the current enumeration value is provided via the *Value Attribute* of the *MultiStateValueDiscrete Variable*. The *ValueAsText Property* provides the localized text representation of the enumeration value. It can be used by *Clients* only interested in displaying the text to subscribe to the *Property* instead of the *Value Attribute*.

5.3.4 ArrayItemType

5.3.4.1 General

This abstract *VariableType* defines the general characteristics of an *ArrayItem*. Values are exposed in an array, but the content of the array represents a single entity like an image. Other *DataItems* might contain arrays that represent for example several values of several temperature sensors of a boiler.

ArrayItemType or its subtype shall only be used when the *Title* and *AxisScaleType Properties* can be filled with reasonable values. If this is not the case *DataItemType* and subtypes like *AnalogItemType,* which also support arrays, shall be used. The *ArrayItemType* is formally defined in Table 10.

Table 10 – ArrayItemType definition

Attribute	**Value**				
BrowseName	ArrayItemType				
IsAbstract	True				
ValueRank	0 (0 = OneOrMoreDimensions)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *DataItemType* defined in 5.3.1; i.e the *Properties* of that type are inherited.					
HasSubtype	VariableType	YArrayItemType	Defined in 5.3.4.2		
HasSubtype	VariableType	XYArrayItemType	Defined in 5.3.4.3		
HasSubtype	VariableType	ImageItemType	Defined in 5.3.4.4		
HasSubtype	VariableType	CubeItemType	Defined in 5.3.4.5		
HasSubtype	VariableType	NDimensionArrayItemType	Defined in 5.3.4.6		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Mandatory
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory
HasProperty	Variable	Title	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	AxisScaleType	AxisScaleEnumeration	PropertyType	Mandatory

InstrumentRange defines the range of the *Value* of the *ArrayItem.*

EURange defines the value range of the ArrayItem likely to be obtained in normal operation. It is intended for such use as automatically scaling a bar graph display.

EngineeringUnits holds the information about the engineering units of the *Value* of the *ArrayItem*.

For additional information about *InstrumentRange*, *EURange*, and *EngineeringUnits* see the description of *AnalogItemType* in 5.3.2.

Title holds the user readable title of the *Value* of the *ArrayItem*.

AxisScaleType defines the scale to be used for the axis where the *Value* of the *ArrayItem* shall be displayed.

The *StatusCode SemanticsChanged* bit shall be set if any of the *InstrumentRange*, *EURange*, *EngineeringUnits* or *Title Properties* are changed (see 5.2 for additional information).

5.3.4.2 YArrayItemType

YArrayItemType represents a single-dimensional array of numerical values used to represent spectra or distributions where the x axis intervals are constant. *YArrayItemType* is formally defined in Table 11.

Table 11 – YArrayItemType definition

Attribute	**Value**				
BrowseName	YArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	BaseDataType				
ArrayDimensions	{0} (0 = UnknownSize)				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

The *Value* of the *YArrayItem* contains the numerical values for the Y-Axis. *Engineering Units* and *Range* for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* and *DoubleComplexNumberType*.

The *XAxisDefinition Property* holds the information about the *Engineering Units* and *Range* for the X-Axis.

The *StatusCode SemanticsChanged* bit shall be set if any of the following five *Properties* are changed: *InstrumentRange, EURange, EngineeringUnits, Title* or *XAxisDefinition* (see 5.2 for additional information).

Figure 3 shows an example of how *Attributes* and *Properties* may be used in a graphical interface.

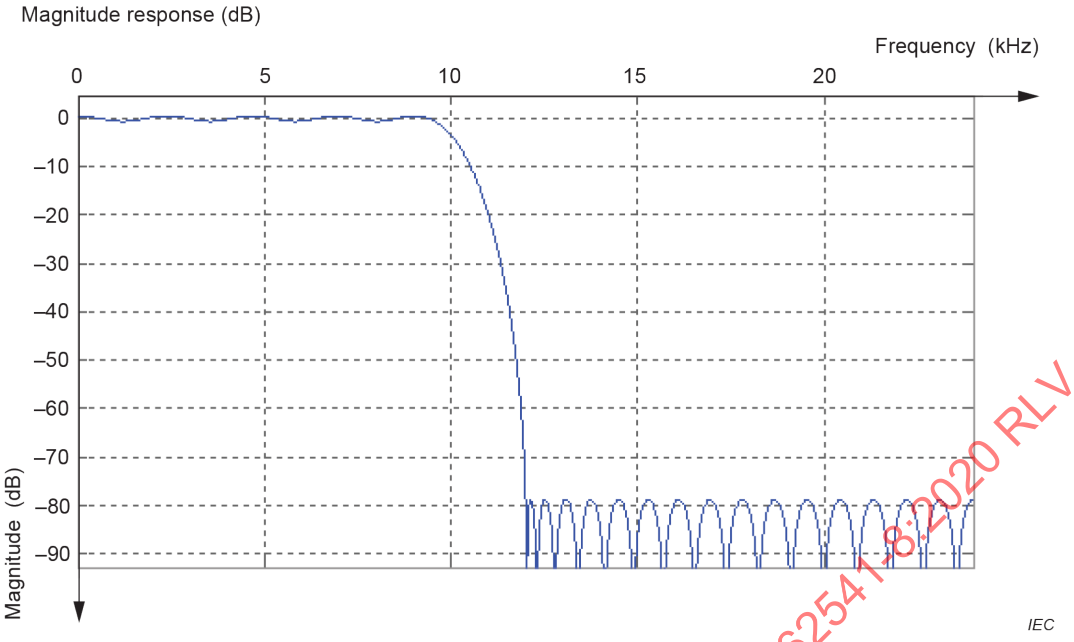


Figure 3 – Graphical view of a *YArrayItem*

Table 12 describes the values of each element presented in Figure 3.

Table 12 – *YArrayItem* item description

Attribute / Property	Item value
Description	Magnitude Response (dB)
axisScaleType	AxisScaleEnumeration.LINEAR_0
InstrumentRange.low	-90
InstrumentRange.high	5
EURange.low	-90
EURange.high	2
EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
EngineeringUnits.unitId	2N
EngineeringUnits.displayName	"en-us", "dB"
EngineeringUnits.description	"en-us", "decibel"
Title	Magnitude
XAxisDefinition.EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
XAxisDefinition.EngineeringUnits.unitId	kHz
XAxisDefinition.EngineeringUnits.displayName	"en-us", "kHz"
XAxisDefinition.EngineeringUnits.description	"en-us", "kilohertz"
XAxisDefinition.Range.low	0
XAxisDefinition.Range.high	25
XAxisDefinition.title	"en-us", "Frequency"
XAxisDefinition.axisScaleType	AxisScaleEnumeration.LINEAR_0
XAxisDefinition.axisSteps	null
Interpretation notes: • Not all elements of this table are used in Figure 3. • The X axis is displayed in reverse order, however, the *XAxisDefinition.Range.low* shall be lower than *XAxisDefinition.Range.high*. It is only a graphical representation that reverses the display order. • There is a constant X axis.	

5.3.4.3 XYArrayItemType

XYArrayItemType represents a vector of XVType values like a list of peaks, where XVType.x is the position of the peak and XVType.value is its intensity. *XYArrayItemType* is formally defined in Table 13.

Table 13 – XYArrayItemType definition

Attribute	Value				
BrowseName	XYArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	XVType (defined in 5.6.8)				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

The *Value* of the *XYArrayItem* contains an array of structures (XVType) where each structure specifies the position for the X-Axis (XVType.x) and the value itself (XVType.value), used for the Y-Axis. Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

XAxisDefinition Property holds the information about the *Engineering Units* and *Range* for the X-Axis.

The *axisSteps* of *XAxisDefinition* shall be set to NULL because it is not used.

The *StatusCode SemanticsChanged* bit shall be set if any of the *InstrumentRange, EURange, EngineeringUnits, Title* or *XAxisDefinition Properties* are changed (see 5.2 for additional information).

5.3.4.4 ImageItemType

ImageItemType defines the general characteristics of an ImageItem which represents a matrix of values like an image, where the pixel position is given by X which is the column and Y the row. The value is the pixel intensity.

ImageItemType is formally defined in Table 14.

Table 14 – ImageItemType definition

Attribute	**Value**				
BrowseName	ImageItemType				
IsAbstract	False				
ValueRank	2 (2 = two dimensional array)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, ComplexNumberType and DoubleComplexNumberType.

The *ArrayDimensions Attribute* for *Variables* of this type or subtypes shall use the first entry in the array ([0]) to define the number of columns and the second entry ([1]) to define the number of rows, assuming the size of the matrix is not dynamic.

XAxisDefinition Property holds the information about the engineering units and range for the X-Axis.

YAxisDefinition Property holds the information about the engineering units and range for the Y-Axis.

The *StatusCode.SemanticsChanged* bit shall be set if any of the *InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition* or *YAxisDefinition Properties* are changed.

5.3.4.5 CubeItemType

CubeItemType represents a cube of values like a spatial particle distribution, where the particle position is given by X which is the column, Y the row and Z the depth. In the example of a spatial partical distribution, the value is the particle size. *CubeItemType* is formally defined in Table 15.

Table 15 – CubeItemType definition

Attribute	**Value**				
BrowseName	CubeItemType				
IsAbstract	False				
ValueRank	3 (3 = three dimensional array)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	ZAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Engineering units and range for the *Value* are defined by corresponding *Properties* inherited from the *ArrayItemType*.

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* and *DoubleComplexNumberType*.

The *ArrayDimensions Attribute* for *Variables* of this type or subtypes should use the first entry in the array ([0]) to define the number of columns, the second entry ([1]) to define the number of rows, and the third entry ([2]) define the number of steps in the Z axis, assuming the size of the matrix is not dynamic.

XAxisDefinition Property holds the information about the engineering units and range for the X-Axis.

YAxisDefinition Property holds the information about the engineering units and range for the Y-Axis.

ZAxisDefinition Property holds the information about the engineering units and range for the Z-Axis.

The *StatusCode SemanticsChanged* bit shall be set if any of *the InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition, YAxisDefinition* or *ZAxisDefinition Properties* are changed (see 5.2 for additional information).

5.3.4.6 NDimensionArrayItemType

This *VariableType* defines a generic multi-dimensional *ArrayItem*.

This approach minimizes the number of types however it may be proved more difficult to utilize for control system interactions.

NDimensionArrayItemType is formally defined in Table 16.

Table 16 – NDimensionArrayItemType definition

Attribute	**Value**				
BrowseName	NdimensionArrayItemType				
IsAbstract	False				
ValueRank	0 (0 = OneOrMoreDimensions)				
DataType	BaseDataType				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *ArrayItemType* defined in 5.3.4.1					
HasProperty	Variable	AxisDefinition	AxisInformation []	PropertyType	Mandatory

The *DataType* of this *VariableType* is restricted to SByte, Int16, Int32, Int64, Float, Double, ComplexNumberType and DoubleComplexNumberType.

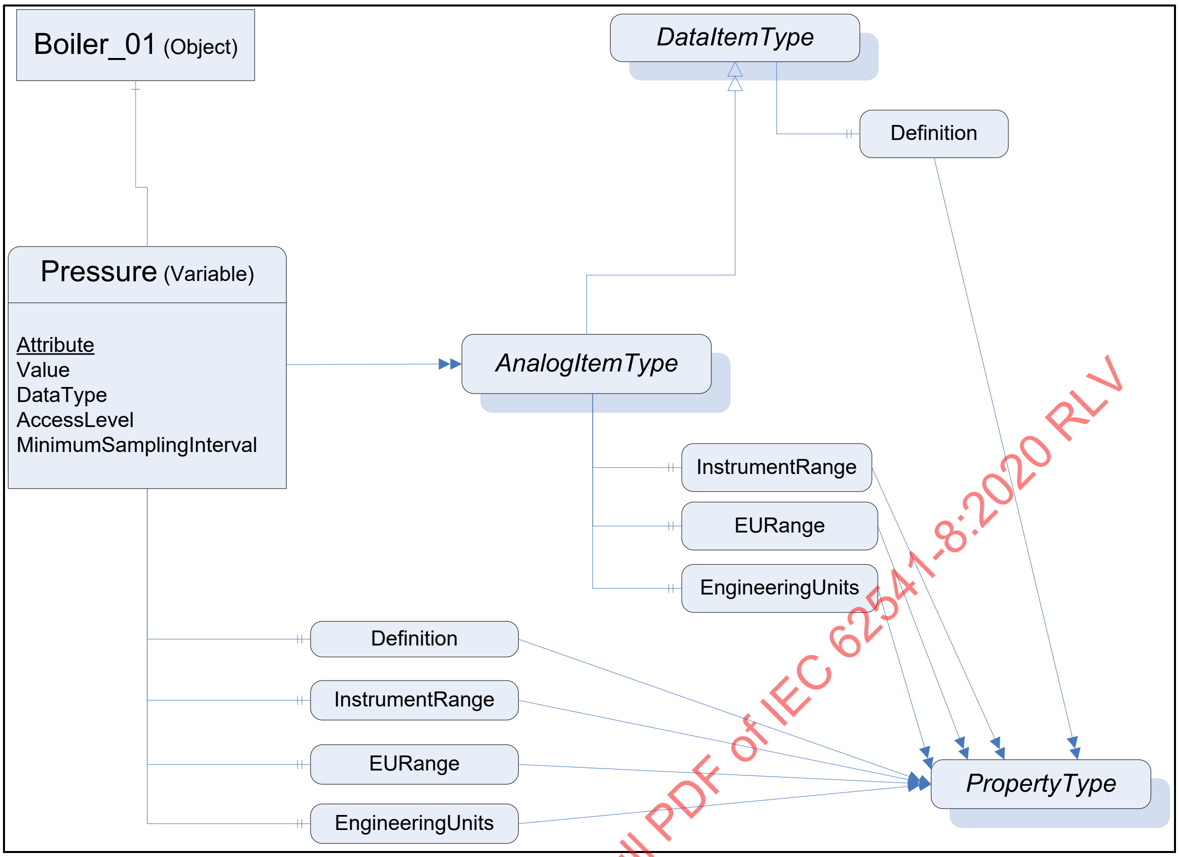
AxisDefinition Property holds the information about the *Engineering Units* and *Range* for all axis.

The *StatusCode SemanticsChanged* bit shall be set if any of *the InstrumentRange, EURange, EngineeringUnits, Title* or *AxisDefinition Properties* are changed (see 5.2 for additional information).

5.4 Address Space model

*DataItem*s are always defined as data components of other *Node*s in the *AddressSpace*. They are never defined by themselves. A simple example of a container for *DataItem*s would be a "Folder Object" but it can be an *Object* of any other type.

Figure 4 illustrates the basic *AddressSpace* model of a *DataItem*, in this case an *AnalogItem*.



IEC

Figure 4 – Representation of DataItems in the AddressSpace

Each *DataItem* is represented by a *DataVariable* with a specific set of *Attributes*. The *TypeDefinition* reference indicates the type of the *DataItem* (in this case the *AnalogItemType*). Additional characteristics of *DataItems* are defined using *Properties*. The *VariableTypes* in 5.2 specify which properties may exist. These *Properties* have been found to be useful for a wide range of Data Access clients. *Servers* that want to disclose similar information should use the OPC-defined *Property* rather than one that is vendor-specific.

The above figure shows only a subset of *Attributes* and *Properties*. Other *Attributes* that are defined for *Variables* in IEC 62541-3 (e.g., *Description*) may also be available.

5.5 Attributes of DataItems

This subclause lists the *Attributes* of *Variables* that have particular importance for Data Access. They are specified in detail in IEC 62541-3. The following *Attributes* are particularly important for Data Access:

- Value,
- DataType,
- AccessLevel,
- MinimumSamplingInterval.

Value is the most recent value of the *Variable* that the *Server* has. Its data type is defined by the *DataType Attribute*. The *AccessLevel Attribute* defines the *Server's* basic ability to access current data and *MinimumSamplingInterval* defines how current the data is.

When a client requests the *Value Attribute* for reading or monitoring, the *Server* will always return a *StatusCode* (the quality and the *Server's* ability to access/provide the value) and, optionally, a *ServerTimestamp* and/or a *SourceTimestamp* – based on the *Client's* request. See IEC 62541-4 for details on *StatusCode* and the meaning of the two timestamps. Specific status codes for Data Access are defined in 6.3.

5.6 DataTypes

5.6.1 Overview

Following is a description of the *DataTypes* defined in this specification.

DataTypes like *String, Boolean*, *Double* or *LocalizedText* are defined in IEC 62541-3. Their representation is specified in IEC 62541-5.

5.6.2 Range

This structure defines the *Range* for a value. Its elements are defined in Table 17.

Table 17 – *Range* DataType structure

Name	Type	Description
Range	structure	
low	Double	Lowest value in the range.
high	Double	Highest value in the range.

If the *DataType* of the *Variable* is Int64 or UInt64 not all values can be covered with the *Range DataType*. In that case, the next lowest respectively the next highest value shall be used.

If a limit is not known a NaN shall be used.

Its representation in the *AddressSpace* is defined in Table 18.

Table 18 – *Range* definition

Attributes	Value
BrowseName	Range

5.6.3 EUInformation

This structure contains information about the *EngineeringUnits*. Its elements are defined in Table 19.

Table 19 – *EUInformation* DataType structure

Name	Type	Description
EUInformation	structure	
namespaceUri	String	Identifies the organization (company, standards organization) that defines the *EUInformation*.
unitId	Int32	Identifier for programmatic evaluation. −1 is used if a *unitId* is not available.
displayName	LocalizedText	The *displayName* of the engineering unit is typically the abbreviation of the engineering unit, for example "h" for hour or "m/s" for meter per second.
description	LocalizedText	Contains the full name of the engineering unit such as "hour" or "meter per second".

Its representation in the *AddressSpace* is defined in Table 20.

Table 20 – *EUInformation* definition

Attributes	Value
BrowseName	EUInformation

To facilitate interoperability, OPC UA specifies how to apply the widely accepted "Codes for Units of Measurement" published by the "United Nations Centre for Trade Facilitation and Electronic Business" (see UN/CEFACT: UNECE Recommendation N° 20). It uses and is based on the International System of Units (SI Units) but in addition provides a fixed code that can be used for automated evaluation. This recommendation has been accepted by many industries on a global basis.

The UNECE recommendation can be found here:

https://www.unece.org/cefact/codesfortrade/codes_index.html

The latest UNECE version (Rev 12. Filename = rec20_Rev12e_2016.xls, published in 2016) is available here:

http://www.unece.org/fileadmin/DAM/cefact/recommendations/rec20/rec20_Rev12e_2016.xls

The mapping of the UNECE codes to OPC UA (EUInformation.unitId) is available here:

http://www.opcfoundation.org/UA/EngineeringUnits/UNECE/UNECE_to_OPCUA.csv

Table 21 contains a small excerpt of the published Annex with Code Lists:

Table 21 – Examples from UNECE Recommendation N° 20

Excerpt from Recommendation N°. 20, Annex 1			
Common Code	**Name**	**Conversion Factor**	**Symbol**
C81	radian		rad
C25	milliradian	10^{-3} rad	mrad
MMT	millimetre	10^{-3} m	mm
HMT	hectometre	10^{2} m	hm
KTM	kilometre	10^{3} m	km
KMQ	kilogram per cubic metre	kg/m^3	kg/m^3
FAH	degree Fahrenheit	5/9 × K	° F
J23	degree Fahrenheit per hour	1,543 210 × 10^{-4} K/s	° F/h

Specific columns of this table shall be used to create the *EUInformation* structure as defined by the following rules:

- The Common Code is represented as an alphanumeric variable length of 3 characters. It shall be used for the *EUInformation.unitId*. The following pseudo code specifies the algorithm to convert the Common Code into an Int32 as needed for *EUInformation.unitId*:

```
Int32 unitId = 0;
Int32 c;
for (i=0; i<=3;i++)
{
     c = CommonCode[i];
     if (c == 0) break;          // end of Common Code
     unitId = unitId << 8;
     unitId = unitId | c;
}
```

- The Symbol field shall be copied to the *EUInformation.displayName*. The localeId field of *EUInformation.displayName* shall be empty.
- The Name field shall be used for *EUInformation.description*. If the name is copied, then the localeId field of *EUInformation.description* shall be empty. If the name is localized, then the localeId field shall specify the correct locale.

The *EUInformation.namespaceUri* shall be http://www.opcfoundation.org/UA/units/un/cefact.

It is advantegous to use Recommendation N° 20 as specified, because it can be programmatically interpreted by generic OPC UA *Clients*. However, the *EUInformation* structure has been defined such that other standards bodies can incorporate their engineering unit definitions into OPC UA. If *Servers* use such an approach, then they shall identify this standards body by using a proper *namespaceUri* in *EUInformation.namespaceUri*.

5.6.4 ComplexNumberType

This structure defines float IEEE 32 bits complex value. Its elements are defined in Table 22.

Table 22 – ComplexNumberType DataType structure

Name	Type	Description
ComplexNumberType	structure	
real	Float	Value real part
imaginary	Float	Value imaginary part

Its representation in the *AddressSpace* is defined in Table 23.

Table 23 – ComplexNumberType definition

Attributes	Value
BrowseName	ComplexNumberType

5.6.5 DoubleComplexNumberType

This structure defines double IEEE 64 bits complex value. Its elements are defined in Table 24.

Table 24 – DoubleComplexNumberType DataType structure

Name	Type	Description
DoubleComplexNumberType	structure	
real	Double	Value real part
imaginary	Double	Value imaginary part

Its representation in the *AddressSpace* is defined in Table 25.

Table 25 – DoubleComplexNumberType definition

Attributes	**Value**
BrowseName	DoubleComplexNumberType

5.6.6 AxisInformation

This structure defines the information for auxiliary axis for *ArrayItemType Variable*s.

There are three typical uses of this structure:

a) the step between points is constant and can be predicted using the range information and the number of points. In this case, *axisSteps* can be set to NULL;

b) the step between points is not constant, but remains the same for a long period of time (from acquisition to acquisition for example). In this case, *axisSteps* contains the value of each step on the axis;

c) the step between points is not constant and changes at every update. In this case, a type like *XYArrayType* shall be used and *axisSteps* is set to NULL.

Its elements are defined in Table 26.

Table 26 – AxisInformation DataType structure

Name	Type	Description
AxisInformation	structure	
engineeringUnits	EUInformation	Holds the information about the engineering units for a given axis.
eURange	Range	Limits of the range of the axis
title	Localizedtext	User readable axis title, useful when the units are %, the Title may be "Particle size distribution"
axisScaleType	AxisScaleEnumeration	LINEAR, LOG, LN, defined by AxisSteps
axisSteps	Double[]	Specific value of each axis steps, may be set to "Null" if not used

When the steps in the axis are constant, *axisSteps* may be set to "Null" and in this case, the *Range* limits are used to compute the steps. The number of steps in the axis comes from the parent *ArrayItem.ArrayDimensions*.

5.6.7 AxisScaleEnumeration

This enumeration identifies on which type of axis the data shall be displayed. Its values are defined in Table 27.

Table 27 – AxisScaleEnumeration values

Value	Description
LINEAR_0	Linear scale
LOG_1	Log base 10 scale
LN_2	Log base e scale

Its representation in the *AddressSpace* is defined in Table 28.

Table 28 – AxisScaleEnumeration definition

Attributes	Value
BrowseName	AxisScaleEnumeration

5.6.8 XVType

This structure defines a physical value relative to a X axis and it is used as the *DataType* of the Value of *XYArrayItemType*. For details see 5.3.4.3.

Many devices can produce values that can perfectly be represented with a float IEEE 32 bits but, they can position them on the X axis with an accuracy that requires double IEEE 64 bits. For example, the peak value in an absorbance spectrum where the amplitude of the peak can be represented by a float IEEE 32 bits, but its frequency position required 10 digits which implies the use of a double IEEE 64 bits.

Its elements are defined in Table 29.

Table 29 – XVType DataType structure

Name	Type	Description
XVType	structure	
x	Double	Position on the X axis of this value
value	Float	The value itself

Its representation in the *AddressSpace* is defined in Table 30.

Table 30 – XVType definition

Attributes	Value
BrowseName	XVType

6 Data Access specific usage of Services

6.1 General

IEC 62541-4 specifies the complete set of services. The services needed for the purpose of DataAccess are:

- The *View* service set and *Query* service set to detect *DataItem*s, and their *Properties*.
- The *Attribute* service set to read or write *Attribute*s and in particular the value *Attribute*.
- The *MonitoredItem* and *Subscription* service set to set up monitoring of *DataItem*s and to receive data change notifications.

6.2 PercentDeadband

The *DataChangeFilter* in IEC 62541-4 defines the conditions under which a data change notification shall be reported. This filter contains a *deadbandValue* which can be of type *AbsoluteDeadband* or *PercentDeadband*. IEC 62541-4 already specifies the behaviour of the *AbsoluteDeadband*. This sub-clause specifies the behaviour of the *PercentDeadband* type.

DeadbandType = PercentDeadband

For this type of deadband the *deadbandValue* is defined as the percentage of the *EURange*. That is, it applies only to *AnalogItems* with an *EURange Property* that defines the typical value range for the item. This range shall be multiplied with the *deadbandValue* and then compared to the actual value change to determine the need for a data change notification. The following pseudo code shows how the deadband is calculated:

```
DataChange if (absolute value of (last cached value - current value) >
                (deadbandValue/100.0) * ((high-low) of EURange)))
```

The range of the *deadbandValue* is from 0,0 to 100,0 per cent. Specifying a *deadbandValue* outside of this range will be rejected and reported with the *StatusCode* Bad_DeadbandFilterInvalid (see Table 31).

If the Value of the *MonitoredItem* is an array, then the deadband calculation logic shall be applied to each element of the array. If an element that requires a DataChange is found, then no further deadband checking is necessary and the entire array shall be returned.

6.3 Data Access status codes

6.3.1 Overview

This subclause defines additional codes and rules that apply to the *StatusCode* when used for Data Access values.

The general structure of the *StatusCode* is specified in IEC 62541-4 and includes a set of common operational result codes that also apply to Data Access.

6.3.2 Operation level result codes

Certain conditions under which a *Variable* value was generated are only valid for automation data and in particular for device data; they are similar, but are slightly more generic than the description of data quality in the various fieldbus specifications.

Table 31 contains codes with BAD severity which indicates a failure.

Table 32 contains codes with UNCERTAIN severity which indicates that the value has been generated under sub-normal conditions.

Table 33 contains GOOD (success) codes.

Note again, that these are the codes that are specific for Data Access and supplement the codes that apply to all types of data which are defined in IEC 62541-4.

Table 31 – Operation level result codes for BAD data quality

Symbolic Id	Description
Bad is defined in IEC 62541-4. It shall be used when there is no special reason why the Value is bad.	
Bad_ConfigurationError	There is a problem with the configuration that affects the usefulness of the value.
Bad_NotConnected	The variable should receive its value from some data source, but has never been configured to do so.
Bad_DeviceFailure	There has been a failure in the device/data source that generates the value that has affected the value.
Bad_SensorFailure	There has been a failure in the sensor from which the value is derived by the device/data source. The limits bits are used to define if the limits of the value have been reached.
Bad_NoCommunication is defined in IEC 62541-4. It shall be used when communications to the data source is defined, but not established, and there is no last known value available.	
Bad_OutOfService	The source of the data is not operational.
~~Bad_LastKnown~~	OPC UA requires that the *Server* shall return a Null value when the *Severity* is Bad. Therefore, the Fieldbus code "Bad_LastKnown" shall be mapped to Uncertain_NoCommunicationLastUsable.
Bad_DeadbandFilterInvalid	The specified *PercentDeadband* is not between 0.0 and 100.0 or a *PercentDeadband* is not supported, since an *EURange* is not configured.
Bad_WaitingForInitialData is defined in IEC 62541-4.	

Table 32 – Operation level result codes for UNCERTAIN data quality

Symbolic Id	Description
Uncertain is defined in IEC 62541-4. It shall be used when there is no special reason why the Value is uncertain.	
Uncertain_ NoCommunicationLastUsable	Communication to the data source has failed. The variable value is the last value that had a good quality and it is uncertain whether this value is still current. The server timestamp in this case is the last time that the communication status was checked. The time at which the value was last verified to be true is no longer available.
Uncertain_ LastUsableValue	Whatever was updating this value has stopped doing so. This happens when an input variable is configured to receive its value from another variable and this configuration is cleared after one or more values have been received. This status/substatus is not used to indicate that a value is stale. Stale data can be detected by the client looking at the timestamps.
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. This status/substatus is set only during configuration while the variable is not operational (while it is out-of-service).
Uncertain_ SensorNotAccurate	The value is at one of the sensor limits. The Limits bits define which limit has been reached. Also set if the device can determine that the sensor has reduced accuracy (e.g. degraded analyzer), in which case the Limits bits indicate that the value is not limited.
Uncertain_ EngineeringUnitsExceeded	The value is outside of the range of values defined for this parameter. The Limits bits indicate which limit has been reached or exceeded.
Uncertain_SubNormal	The value is derived from multiple sources and has less than the required number of Good sources.

Table 33 – Operation level result codes for GOOD data quality

Symbolic Id	Description
Good is defined in IEC 62541-4. It shall be used when there are no special conditions.	
Good_LocalOverride	The value has been Overridden. Typically, this means the input has been disconnected and a manually-entered value has been "forced".

6.3.3 LimitBits

The bottom 16 bits of the *StatusCode* are bit flags that contain additional information, but do not affect the meaning of the *StatusCode*. Of particular interest for *DataItem*s is the *LimitBits* field. In some cases, such as sensor failure it can provide useful diagnostic information.

Servers that do not support Limit have to set this field to 0.

Annex A
(informative)

OPC COM DA to UA mapping

A.1 Overview

This Annex provides details on mapping OPC COM Data Access (DA) information to OPC UA to help vendors migrate to OPC UA based systems while still being able to access information from existing OPC COM DA systems.

The OPC Foundation provides COM UA Wrapper and Proxy samples that act as a bridge between the OPC DA and the OPC UA systems.

The COM UA Wrapper is an OPC UA Server that wraps an OPC DA Server and with that enables an OPC UA Client to access information from the DA Server. The COM UA Proxy enables an OPC DA Client to access information from an OPC UA Server.

The mappings describe generic DA interoperability components. It is recommended that vendors use this mapping if they develop their own components, however, some applications may benefit from vendor-specific mappings.

A.2 Security considerations

COM DA relies on the Microsoft COM security infrastructure and does not specify any security parameters such as user identity. The developer of UA Wrapper and Proxy therefore has to consider the mapping of security aspects.

The COM UA Wrapper for instance may accept any Username/password and then try to impersonate this user by calling proper Windows services before connecting to the COM DA Server.

A.3 COM UA wrapper for OPC DA Server

A.3.1 Information Model mapping

A.3.1.1 General

OPC DA defines 3 elements in the address space: Branch, Item and Property. The COM UA Wrapper maps these types to the OPC UA types as described in A.3.1.2 to A.3.1.4.

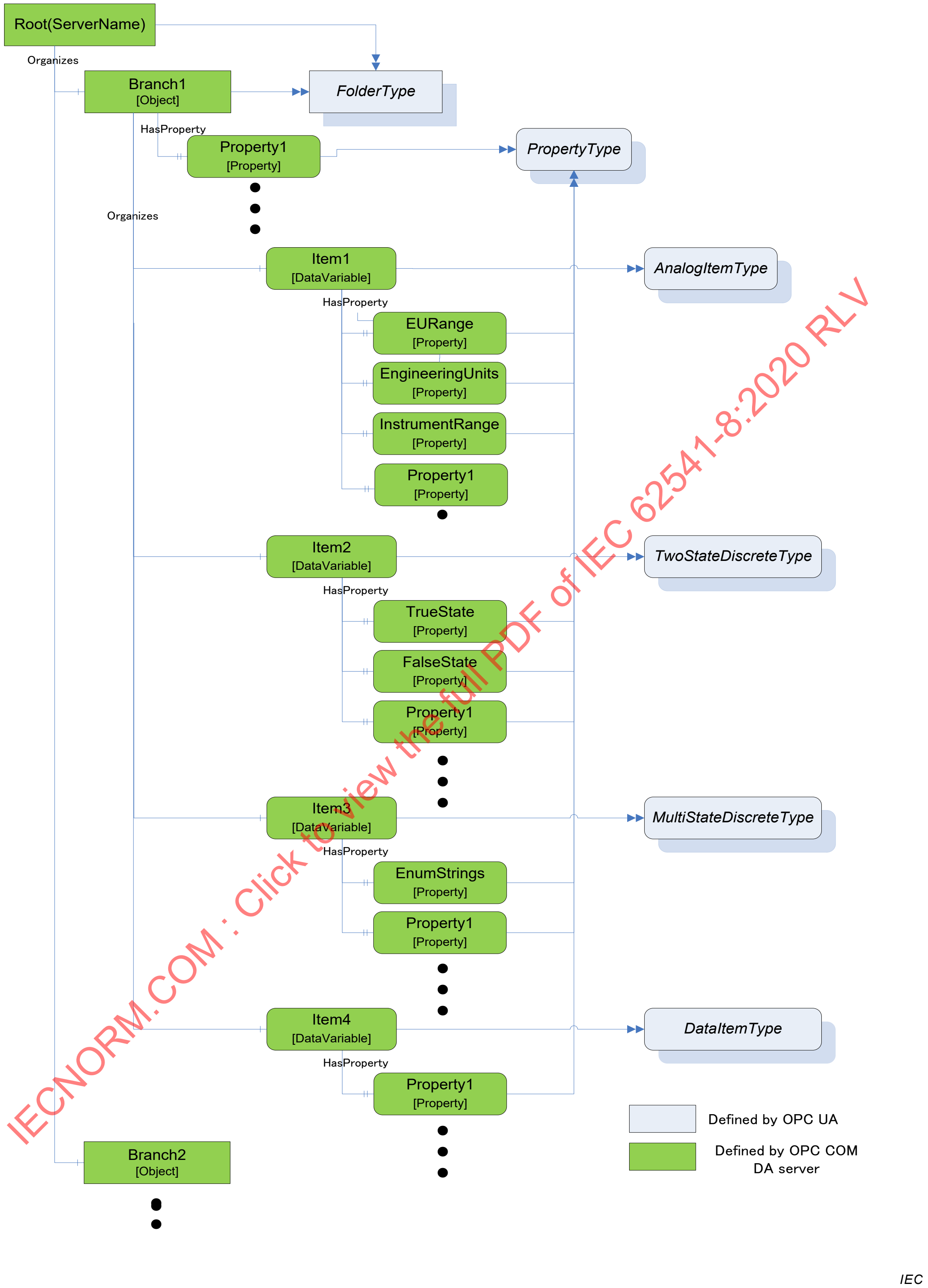


Figure A.1 – Sample OPC UA Information Model for OPC DA

A.3.1.2 Branch

DA Branches are represented in the COM UA Wrapper as *Objects* of *FolderType*.

The top-level branch (the root) should be represented by an *Object* where the *BrowseName* is the Server ProgId.

The OPC DA Address space hierarchy is discovered using the ChangeBrowsePosition from the Root and BrowseOPCItemIds to get the Branches, Items and Properties.

The name returned from the BrowseOPCItemIds enumString is used as the BrowseName and the DisplayName for each Branch. See also A.3.1.5.

The ItemId obtained using the GetItemID is used as a part of the NodeId for each Branch. See also A.3.1.5.

An OPC UA *Folder* representing a DA Branch uses the *Organizes References* to reference child DA Branches and uses *HasComponent References* for DA Leafs (Items). It is acceptable for customized wrappers to use a sub-type of these ReferenceTypes.

A.3.1.3 Item

DA items (leafs) are represented in the COM UA Wrapper as *Variables*. The VariableType depends on the existance of special DA properties as follows:

- *AnalogItemType*: An item in the DA server that has High EU and Low EU properties or its EU Type property is Analog is represented as *Variable* of *AnalogItemType* in the COM UA Wrapper. *The AnalogItemType* has the following *Properties:*
  - *EURange*: The values of the High EU and Low EU properties of the DA Item are assigned to the *EURange Property*
  - *EngineeringUnits*: The value of the Engineering Unit property of the DA Item are assigned to the *EngineeringUnits Property*.
  - InstrumentRange: The values of the High IR and Low IR properties of the DA Item are assigned to the InstrumentRange Property
- *TwoStateDiscreteType: An* item in DA server that has Open Label and Close Label properties is represented as *Variable of TwoStateDiscreteType* in the COM UA Wrapper*. The TwoStateDiscreteType* has the following *Properties*
  - *TrueState*: The value of the Close Label property of the DA item is assigned to the *TrueState Property*.
  - FalseState: The value of the Open Label property of the DA item is assigned to the FalseState Property.
- *MultiStateDiscreteType***:** An item in the DA server that has its EU Type property as enumerated is represented as *Variable* of *MultiStateDiscreteType* in the COM UA Wrapper. The *MultiStateDiscreteType* has the following *Property*:
  - *EnumStrings*: The enumerated values of the EUInfo Property of the DA item are assigned to the *EnumStrings Property*.
- *DataItemType:* An item in the DA Server that is not any of the above types is represented as *Variable* of *DataItemType* in the COM UA Wrapper*.*

Below are mappings that are common for all item types

- The name of the item in the DA Server is used as the *BrowseName* and the *DisplayName* for the *Node* in the COM UA Wrapper. See also clause A.3.1.5.
- The ItemId in the DA server is used as a part of the *NodeId* for the *Node*. See also clause A.3.1.5.
- TimeZone property in the DA server is represented by a *TimeZone Property.*
- The Description property value in the DA server is assigned to the *Description Attribute*.
- *The* DataType property value in the DA server is assigned *to the DataType Attribute*.
- If the item in the DA server is an array, the *ValueRank Attribute* is set as *OneOrMoreDimensions.* If not, it is set to *Scalar.*
- The *AccessLevel Attribute* is set with the AccessRights value in the DA server:

– OPC_READABLE -> Readable

– OPC_WRITABLE -> Writable

Note that the same values are also set for the UserAccessLevel in the COM UA Wrapper.

- *The* ScanRate property value in the DA server is assigned to the *MinimumSamplingInterval Attribute*.

Any *Properties* added to a Node in the COM UA Wrapper are referenced using the *HasProperty ReferenceType*.

A.3.1.4 Property

A property in the DA server is represented in the COM UA Wrapper as a *Variable* with *TypeDefinition* as *PropertyType*.

The properties for an item are retrieved using the QueryAvailableProperties call in the DA server.

Below are mappings of the property details to the OPC UA Property:

- The description of a property in the DA server is used as the *BrowseName* and the *DisplayName* of the Node in the COM UA Wrapper.
- The PropertyID and ItemID (if they exist for the property) in the DA server are used as a part of the *NodeID* for the node in the COM UA Wrapper.
- The DataType value in the DA server is used as value for the *DataType Attribute* of the *Property* in the COM UA Wrapper.
- If the property value in the DA server is an array, the *ValueRank Attribute* of the *Property* is set to *OneOrMoreDimensions.* Otherwise it is set to *Scalar.*
- If the property has an ItemID in the DA server, then the *AccessLevel* attribute for the Node is set to *ReadableOrWriteable*. If not, it is set to *Readable*.

Table A.1 shows the mapping between the common OPC COM DA properties to the OPC UA Node attributes/properties.

Table A.1 – OPC COM DA to OPC UA Properties mapping

Property Name (PropertyID) of OPC COM DA	OPC UA Information Model	OPC UA DataType
Access Rights (5)	AccessLevel Attribute	Int32
EU Units (100)	EngineeringUnits Property	String
Item Description (101)	Description Attribute	String
High EU (102)	EURange Property	Double
Low EU (103)	EURange Property	Double
High Instrument Range (104)	InstrumentRange Property	Double
Low Instrument Range (105)	InstrumentRange Property	Double
Close Label (106)	TrueState Property	String
Open Label (107)	FalseState Property	String
Other Properties (include Vendor specific Properties)	PropertyType	Based on the DataType of the Property

A.3.1.5 BrowseName and DisplayName Mapping

As described above, both the OPC UA Browsename and Displayname for Nodes representing COM DA Branches and Leafs are derived from the name of the corresponding item in the COM DA Server.

This name can only be acquired by using the COM DA Browse Services. In OPC UA, however, the BrowseName and DisplayName are Attributes that Clients can ask for at any time. There are several options to support this in a Wrapper but all of them have pros and cons. Here are some popular implementation options:

a) Allow browsing the complete COM DA Address Space and then build and persist an offline copy of it. Resolve the BrowseName by scanning this offline copy.
   - Pro: the ItemID can be used as is for the OPC UA NodeId.
   - Con: the initial browse can take a while and may have to be repeated for COM DA Servers with a dynamic Address Space.

b) Create OPC UA NodeId values that include both the COM DA ItemID and the Item name. When the OPC UA Client passes such a NodeId to read the BrowseName or DisplayName Attribute, the wrapper can easily extract the name from the NodeId value.
   - Pro: efficient and reliable.
   - Con: the NodeId will not represent the ItemId. It becomes difficult for human users to match the two IDs.

c) A number of COM DA Servers use ItemIDs that consist of a path where the path elements are separated with a delimiter and the last element is the item name. Wrappers may provide ways to configure the delimiter so that they can easily extract the item name.
   - Pro: efficient and reliable. The ItemID can be used as is for the OPC UA NodeId.
   - Con: not a generic solution. Only works for specific COM-DA Servers.

For wrappers that are custom to a specific Server, knowledge of the COM DA server address space can result in other optimizations or short cuts (i.e. the server will always have a certain schema/naming sequence, etc.).

A.3.2 Data and error mapping

A.3.2.1 General

In a DA server, Automation Data is represented by Value, Quality and Time Stamp for a Tag.

The COM UA Wrapper maps the VQT data to the Data Value and Diagnostic Info structures.

The Error codes returned by the DA server are based on the HRESULT type. The COM UA Wrapper maps this error code to an OPC UA Status Code. Figure A.2 illustrates this mapping.

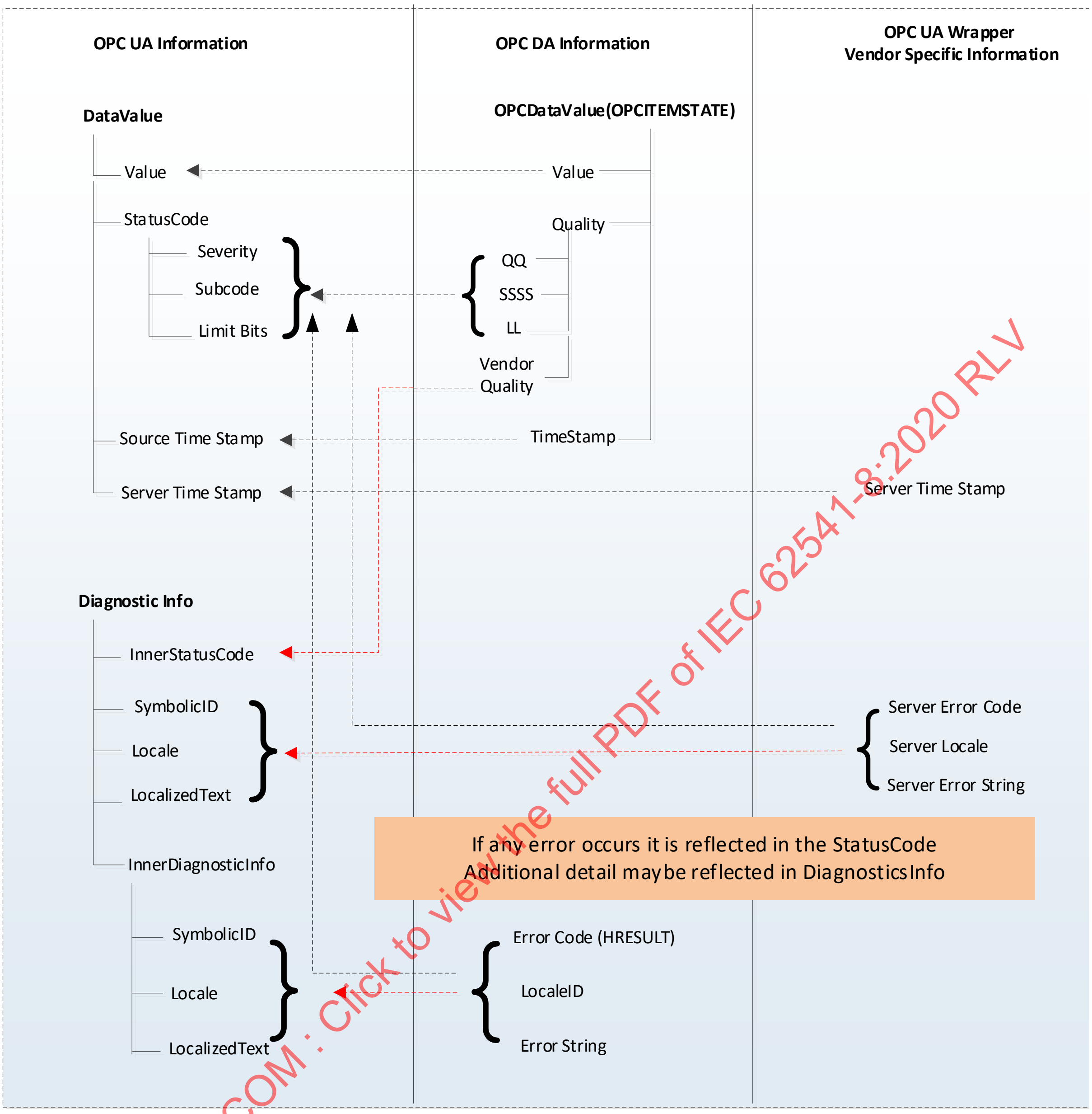


Figure A.2 – OPC COM DA to OPC UA data and error mapping

A.3.2.2 Value

The data values in the DA server are represented as Variant Data type. The COM UA Wrapper converts them to the corresponding OPC UA data type. The mapping is shown in Table A.2.

Table A.2 – DataTypes and mapping

Variant Data Type (In DA server)	OPC UA Data type Mapping in COM UA Server (DataValue structure)
VT_I2	Int16
VT_I4	Int32
VT_R4	Float
VT_R8	Double
VT_BSTR	String
VT_BOOL	Boolean
VT_UI1	Byte
VT_I1	SByte
VT_UI2	UInt16
VT_UI4	UInt32
VT_I8	Int64
VT_UI8	UInt64
VT_DATE	Double
VT_DECIMAL	Decimal
VT_ARRAY	Array of OPC UA types

A.3.2.3 Quality

The Quality of a Data Value in the DA server is represented as a 16-bit value where the lower 8 bits are of the form QQSSSSLL (Q: Main Quality, S: Sub Status, L: Limit) and the higher 8 bits are vendor specific.

The COM UA Wrapper maps the DA server to the OPC UA Status code as shown Figure A.3.

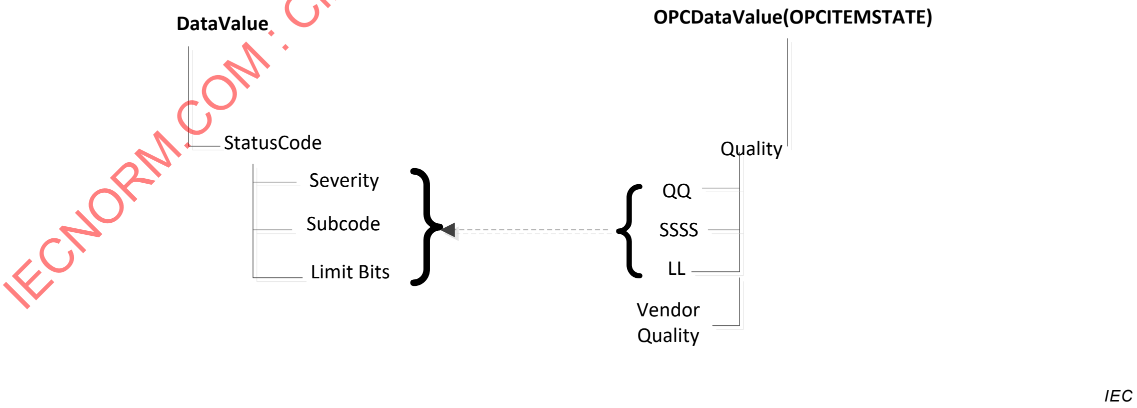


Figure A.3 – Status Code mapping

The primary quality is mapped to the Severity field of the Status code. The Sub Status is mapped to the SubCode and the Limit is mapped to the Limit Bits of the Status Code.

Please note that the Vendor quality is currently discarded.

Table A.3 shows a mapping of the OPC COM DA primary quality mapping to OPC UA status code

Table A.3 – Quality mapping

OPC DA Primary Quality (Quality & Sub status QQSSSS)	OPC UA Status Code
GOOD	Good
LOCAL_OVERRIDE	Good_LocalOverride
UNCERTAIN	Uncertain
SUB_NORMAL	Uncertain_SubNormal
SENSOR_CAL	Uncertain_SensorNotAccurate
EGU_EXCEEDED	Uncertain_EngineeringUnitsExceeded
LAST_USABLE	Uncertain_LastUsableValue
BAD	Bad
CONFIG_ERROR	Bad_ConfigurationError
NOT_CONNECTED	Bad_NotConnected
COMM_FAILURE	Bad_NoCommunication
DEVICE_FAILURE	Bad_DeviceFailure
SENSOR_FAILURE	Bad_SensorFailure
LAST_KNOWN	Bad_OutOfService
OUT_OF_SERVICE	Bad_OutOfService
WAITING_FOR_INITIAL_DATA	Bad_WaitingForInitialData

A.3.2.4 Timestamp

The Timestamp provided for a value in the DA server is assigned to the SourceTimeStamp of the DataValue in the COM UA Wrapper.

The ServerTimeStamp in the DataValue is set to the current time by the COM UA Wrapper at the start of the Read Operation.

A.3.3 Read data

The COM UA Wrapper supports performing Read operations to DA servers of versions 2.05a and 3.

For version 2.05a, the COM UA wrapper creates a Group using the IOPCServer::AddGroup method and adds the items whose data is to be read to the Group using IOPCItemMgmt::AddItems method. The Data is retrieved for the items using the IOPCSyncIO::Read method. The VQT for each item is mapped to the DataValue structure as shown in Figure A.2. Please note that only Read from Device is supported for this version. The "maxAge" parameter is ignored.

For version 3, the COM UA Wrapper uses the IOPCItemIO::Read to retrieve the data. The VQT for each item is mapped to the DataValue structure as shown in Figure A.2. The Read supports both the Read from Device and Cache and uses the "maxAge" parameter.

If there are errors for the items in the Read from the DA server, then these are mapped to the StatusCode of the DataValue in the COM UA Wrapper.

The mapping of the OPC COM DA Read Errors code to OPC UA Status code (in the COM UA Wrapper) is shown in Table A.4:

Table A.4 – OPC DA Read error mapping

OPC DA Error ID	OPC UA Status Code
OPC_E_BADRIGHTS	Bad_NotReadable
E_OUTOFMEMORY	Bad_OutOfMemory
OPC_E_INVALIDHANDLE	Bad_NodeIdUnknown
OPC_E_UNKNOWNITEMID	Bad_NodeIdUnknown
E_INVALIDITEMID	Bad_NodeIdInvalid
E_INVALID_PID	Bad_AttributeIdInvalid
E_ACCESSDENIED	Bad_OutOfService
Others	Bad_UnexpectedError

A.3.4 Write Data

The COM UA Wrapper supports performing Write operations to DA servers of versions 2.05a and 3.

For version 2.05a, the COM UA wrapper creates a Group using the IOPCServer::AddGroup method and adds the items whose data is to be written using IOPCItemMgmt::AddItems method. The value is written for the items using the IOPCSyncIO::Write method. If the StatusCode or TimeStamps (Source or Server) is specified to be written for the item, then the COM UA Wrapper returns a BadWriteNotSupported Status code for the item.

For version 3, the COM UA Wrapper uses the IOPCItemIO::WriteVQT data including StatusCode and TimeStamp.If a SourceTimeStamp is provided, this timestamp is used for the Write else the ServerTimeStamp is used.

If there are errors for the items in the Write from the DA server, then these are mapped to the StatusCode for the corresponding item.

The mapping of the OPC COM DA Write Errors code to OPC UA Status code (in the COM UA Wrapper) is shown in Table A.5.

Table A.5 – OPC DA Write error code mapping

OPC DA Error ID	OPC UA Status Code
E_BADRIGHTS	Bad_NotWritable
DISP_E_TYPEMISMATCH	Bad_TypeMismatch
E_BADTYPE	Bad_TypeMismatch
E_RANGE	Bad_OutOfRange
DISP_E_OVERFLOW	Bad_OutOfRange
E_OUTOFMEMORY	Bad_OutOfMemory
E_INVALIDHANDLE	Bad_NodeIdUnknown
E_UNKNOWNITEMID	Bad_NodeIdUnknown
E_INVALIDITEMID	Bad_NodeIdInvalid
E_INVALID_PID	Bad_NodeIdInvalid
E_NOTSUPPORTED	Bad_WriteNotSupported
S_CLAMP	Good_Clamped
Others	Bad_UnexpectedError

A.3.5 Subscriptions

A subscription is created in the DA server when a MonitoredItem is created in the COM UA Wrapper.

The SamplingInterval and the Deadband value are used for the subscription to setup a periodic data change call back on the COM UA Wrapper. Note that only the PercentDeadbandType is supported by the COM UA Wrapper.

The VQT for each item is mapped to the DataValue structure as shown in Figure A.2 and published to the client by the COM UA Wrapper periodically.

The mapping of the OPC COM DA Read Errors code to OPC UA Status code (in the COM UA Wrapper) is the same as the Read mapping in Figure A.2.

A.4 COM UA proxy for DA Client

A.4.1 Guidelines

The Data Access COM UA Proxy is a COM Server combined with a UA Client. It maps the Data Access address space of UA Data Access Server into the appropriate COM Data Access objects.

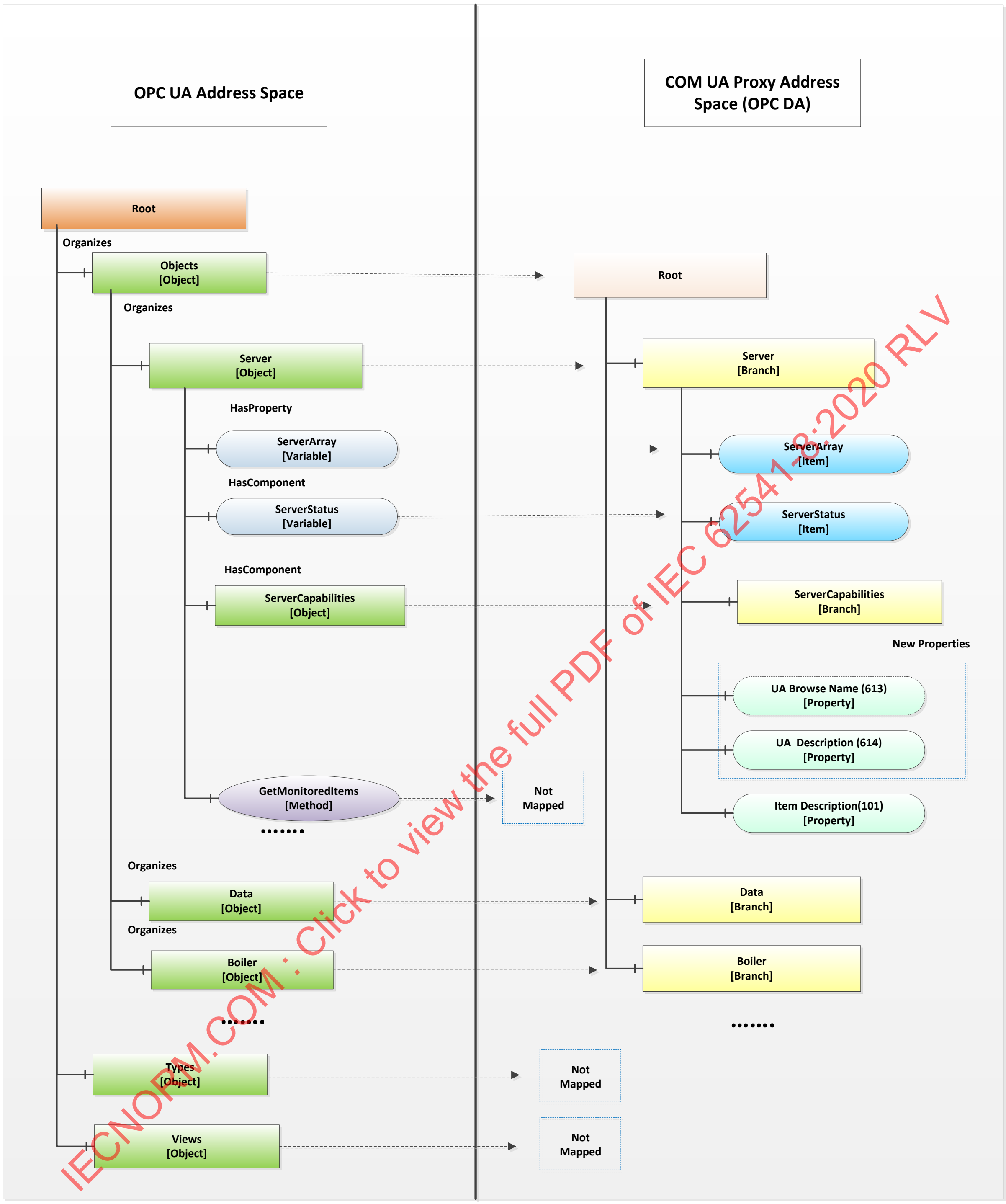
Sublauses A.4.1 through A.4.6 identify the design guidelines and constraints used to develop the Data Access COM UA Proxy provided by the OPC Foundation. In order to maintain a high degree of consistency and interoperability, it is strongly recommended that vendors, who choose to implement their own version of the Data Access COM UA Proxy, follow these same guidelines and constraints.

The Data Access COM Client simply needs to address how to connect to the UA Data Access Server. Connectivity approaches include the one where Data Access COM Clients connect to a UA Data Access Server with a CLSID just as if the target Server were a Data Access COM Server. However, the CLSID can be considered virtual since it is defined to connect to intermediary components that ultimately connect to the UA Data Access Server. Using this approach, the Data Access COM Client calls co-create instance with a virtual CLSID as described above. This connects to the Data Access COM UA Proxy components. The Data Access COM UA Proxy then establishes a secure channel and session with the UA Data Access Server. As a result, the Data Access COM Client gets a COM Data Access Server interface pointer.

A.4.2 Information Model and Address Space mapping

A.4.2.1 General

OPC UA defines 8 Node Class types in the address space Object, Variable, Method, ObjectType, VariableType, ReferenceType, DataType, View. The COM UA Proxy maps only the nodes of Node Class types Object, Variable to the OPC DA types as shown in the figure below. Only the nodes under the Objects node are considered for the COM UA Proxy address space and others such as Types and Views are not mapped. Figure A.4 shows an example mapping of OPC DA to OPC UA information.



IEC

Figure A.4 – Sample OPC DA mapping of OPC UA Information Model and Address Space

A.4.2.2 Object Nodes

A node of Object Node class in the OPC UA server is represented in the Data Access COM UA Proxy as a Branch.

The root of the Data Access COM UA Proxy is the Objects folder of the OPC UA Server.

The OPC UA Address space hierarchy is discovered using the Browse Service for the Objects Node using the following filters:

- BrowseDirection as Forward;
- ReferenceTypeId as Organizes and HasChild;
- IncludeSubtypes as True;
- NodeClassMask as Object and Variable.

The DisplayName of the OPC UA node is used as the Name for each Branch in the Data Access COM UA Proxy

Each Branch in the Data Access COM UA Proxy is assigned 3 properties:

- *UA Browse Name* (Property ID: 613): the value of the *BrowseName* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Description* (Property ID: 614): the value of the *Description* attribute of the node in the OPC UA Server is assigned to this property, if a Description attribute is provided.
- *Item Description* (Property ID: 101): the value of the *DisplayName* attribute of the node in the OPC UA Server is assigned to this property.

NOTE COM DA Clients typically display the ItemID and the Item Description. Since the ItemID generated by the UA Proxy may be particularly difficult to read and understand, proxies may use the DisplayName as value for the Item Description Property as it will be easier to understand by a human user.

A.4.2.3 Variable Nodes

A node of Variable Node class in the OPC UA server is represented in the Data Access COM UA Proxy as an Item.

The DisplayName of the OPC UA node is used as the Name for each Item in the Data Access COM UA Proxy.

The NodeId of the OPC UA node is used as the ItemId for each Item in the Data Access COM UA Proxy, but the ‘=" character is replaced with ‘-’ in the string. For example, NodeId: ns=4,i=10, ItemID = "ns-4;i-10" or NodeId: ns=4,s=FL102, ItemID = "ns-4,s-FL102"

Each Item in the Data Access COM UA Proxy is assigned the following properties based on the node attributes or its references:

Standard Properties:

- *Item Canonical Data Type* (Property ID: 1): the combined value of the *DataType* attribute and the *ValueRank* attribute of the node in the OPC UA Server is assigned to this property (see A.4.3.2).
- *Item Value* (Property ID: 2): the value of the *Value* attribute of the node in the OPC UA Server is assigned to this property. Details on Value mapping are in A.4.3.2.
- *Item Quality* (Property ID: 3): the *StatusCode* of the *Value* obtained for the node in the OPC UA Server is assigned to this property. Details on Quality mapping are in A.4.3.3.
- *Item Timestamp* (Property ID: 4): the *SourceTimestamp or ServerTimestamp* of the *Value* obtained for the node in the OPC UA Server is assigned to this property. Details on Timestamp mapping are in A.4.3.4.
- *Item Access Rights* (Property ID: 5): the value of the *AccessLevel* attribute of the node in the OPC UA Server is assigned to this property based on the following mapping:
  - CurrentRead -> OPC_READABLE
  - CurrentWrite -> OPC_WRITABLE

  The other AccessLevel provided by OPC are ignored
- *Server Scan Rate* (Property ID: 6): the value of the *MinimumSamplingInterval* attribute of the node in the OPC UA Server is assigned to this property.

- *Item EU Type* (Property ID: 7): the EU Type value is assigned based on the references of the node in the OPC UA Server:
  - *Analog(1)*: if the node in the OPC UA Server references a *EURange property* node, then it is assigned the *Analog EU Type.*
  - *Enumerated(2)*: if the node in the OPC UA Server references a *EnumStrings property* node, then it is assigned the *Enumerated EU Type.*
  - *Empty(0)*: For a node in the OPC UA Server that does not meet above criteria, the type is set as 0 (Empty)
- *EU Info* (Property ID: 8): if the node in the OPC UA Server references an *EnumStrings property* node, then the enumerated values of the property node is assigned to this property.
- *EU Units* (Property ID: 100): if the node in the OPC UA Server references a *EngineeringUnits property* node, then the value of the *EngineeringUnits* property node is assigned the *EU Units* property.
- *Item Description* (Property ID: 101): The value of the *DisplayName* attribute of the node in the OPC UA Server is assigned to this property.
- *High EU* (Property ID: 102): if the node in the OPC UA Server references a *EURange property* node, then the *'High'* value of the property node is assigned to this property.
- *Low EU* (Property ID: 103): if the node in the OPC UA Server references a *EURange property* node, then the *'Low'* value of the property node is assigned to this property.
- *High Instrument Range* (Property ID: 104): if the node in the OPC UA Server references an *InstrumentRange property* node, then the *'High'* value of the property node is assigned to this property.
- *Low Instrument Range* (Property ID: 105): if the node in the OPC UA Server references an *InstrumentRange property* node, then the *'Low'* value of the property node is assigned to this property.
- *Contact Close Label* (Property ID: 106): if the node in the OPC UA Server references a *FalseState property* node, then the value of the property node is assigned to this property.
- *Contact Open Label* (Property ID: 107): if the node in the OPC UA Server references a *TrueState property* node, then the value of the property node is assigned to this property.
- *Item Time Zone* (Property ID: 108): if the node in the OPC UA Server references a *TimeZone property* node, then the 'Offset' value of the property node is assigned to this property.

New Properties:

- *UA BuiltIn Type* (Property ID: 610): the identifier value of the *DataType* node associated with the DataType attribute of the node in the OPC UA Server is assigned to this property.
- *UA Data Type Id* (Property ID: 611): the complete NodeId value (namespace and identifier) of the *DataType* node associated with the DataType attribute of the node in the OPC UA Server is assigned to this property.
- *UA Value Rank* (Property ID: 612): the value of the *ValueRank* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Browse Name* (Property ID: 613): the value of the *BrowseName* attribute of the node in the OPC UA Server is assigned to this property.
- *UA Description* (Property ID: 614): the value of the *Description* attribute of the node in the OPC UA Server is assigned to this property.

A.4.2.4 Namespace Indices

For generating ItemIDs, the Proxy uses Namespace Indices. To assure that Clients can persist these ItemIDs, the Namespace Indices shall never change. To accomplish this the Proxy has to persist its Namespace Table and only append entries but never change existing ones.

The Proxy shall also provide a translation from the current Namespace Table in the Server to the persisted Namespace Table.

If you move or copy the Proxy to another machine, the Namespace Table has to be copied to this machine as well.

A.4.3 Data and error mapping

A.4.3.1 General

In an OPC UA Server, Automation Data is represented as a Data Value and and status, in addition additional error data can be provided via Diagnostic Info for a tag

The COM UA Proxy maps the Data Value structure into VQT data and error code.

For successful operations (StatusCode of Good and Uncertain), the COM UA Proxy maps the Status Code of the DataValue to the OPC DA Quality But in case of error (StatusCode of Bad), the Status Code is mapped to the OPC DA Error code.

The StatusCode in the Diagnostic Info returned by the OPC UA Server are mapped to OPC DA Error codes. Figure A.5 illustrates this mapping.

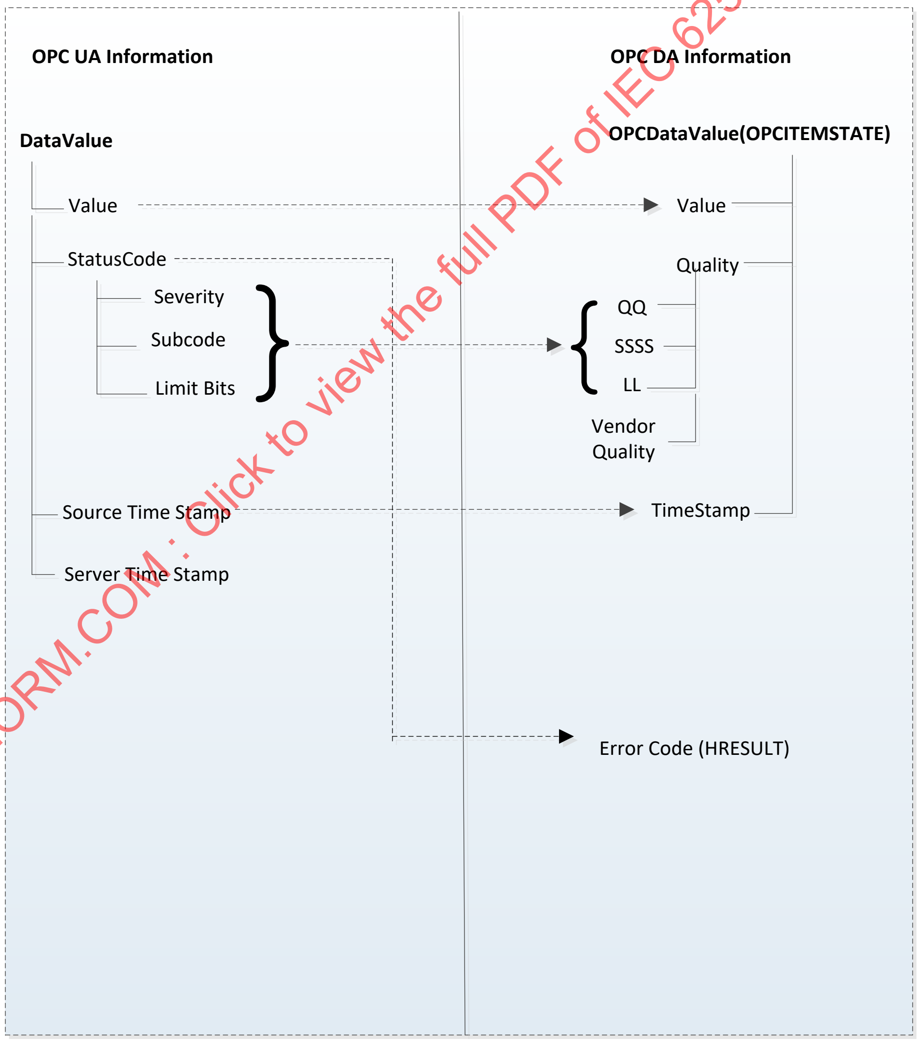


Figure A.5 – OPC UA to OPC DA data & error mapping

A.4.3.2 Value

The COM UA Proxy converts the OPC UA Data Value to the corresponding OPC DA Variant type. The mapping is shown in Table A.6. For DataTypes that are subtypes of an existing base DataType the conversion for the Base DataType is used.

Table A.6 – DataTypes and Mapping

OPC UA Data type (Bin UA Server)	Variant Data Type (In DA server)
Int16	VT_I2
Int32	VT_I4
Float	VT_R4
Double	VT_R8
Decimal	VT_DECIMAL
String	VT_BSTR
Boolean	VT_BOOL
Byte	VT_UI1
SByte	VT_I1
UInt16	VT_UI2
UInt32	VT_UI4
Int64	VT_I8
UInt64	VT_UI8
Guid	VT_BSTR
DateTime	VT_DATE
NodeId	VT_BSTR
XmlElement	VT_BSTR
ExpandedNodeId	VT_BSTR
QualifiedName	VT_BSTR
LocalizedText	VT_BSTR
StatusCode	VT_UI4
ExtensionObject	Array of VT_UI1
Array of above OPC UA types	Array of corresponding Variant type

A.4.3.3 Quality

The Quality of a Data Value in the OPC UA Server is represented as a StatusCode.

The COM UA Proxy maps the Severity, Subcode and the limit bits of the OPC UA Status code to the lower 8 bits of the OPC DA Quality structure (of the form QQSSSSLL). Figure A.6 illustrates this mapping.

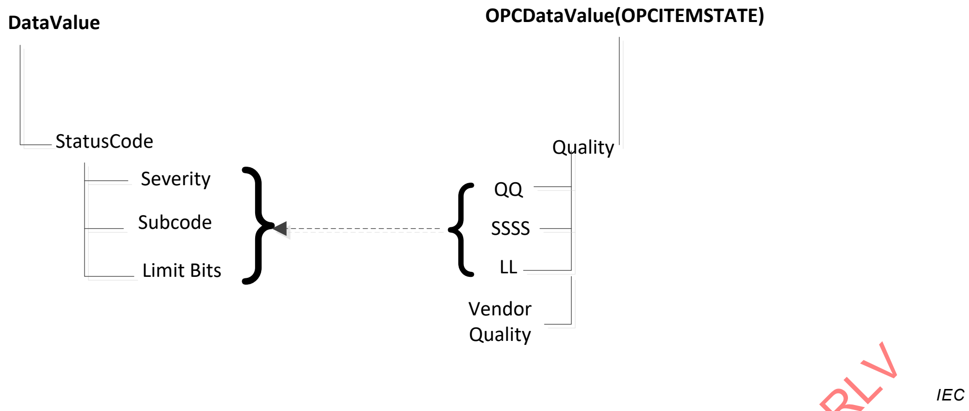


Figure A.6 – OPC UA Status Code to OPC DA quality mapping

The Severity field of the Status code is mapped to the primary quality. The SubCode is mapped to the Sub Status and the Limit Bits are mapped to the Limit field.

Table A.7 shows a mapping of the OPC UA status code to OPC DA primary quality.

Table A.7 – Quality mapping

OPC UA Status Code	OPC DA Primary Quality (Quality & Sub status QQSSSS)
Good	GOOD
Good_LocalOverride	LOCAL_OVERRIDE
Uncertain	UNCERTAIN
Uncertain_SubNormal	SUB_NORMAL
Uncertain_SensorNotAccurate	SENSOR_CAL
Uncertain_EngineeringUnitsExceeded	EGU_EXCEEDED
Uncertain_LastUsableValue	LAST_USABLE
Bad	BAD
Bad_ConfigurationError	CONFIG_ERROR
Bad_NotConnected	NOT_CONNECTED
Bad_NoCommunication	COMM_FAILURE
Bad_OutOfService	OUT_OF_SERVICE
Bad_DeviceFailure	DEVICE_FAILURE
Bad_SensorFailure	SENSOR_FAILURE
Bad_WaitingForInitialData	WAITING_FOR_INITIAL_DATA

A.4.3.4 Timestamp

If available, the SourceTimestamp of the DataValue in the OPC UA Server is assigned to the Timestamp for the value in the COM UA Proxy. If SourceTimestamp is not available, then the ServerTimestamp is used.

A.4.4 Read data

The COM UA Proxy converts all the ItemIds in the Read into valid NodeIds by replacing the '-' with '=' and calls the OPC UA Read Service for the Value Attribute.

If the Read Service call is successful, then DataValue for each node is mapped to the VQT for each item as shown in Figure A.5.

If the Read Service call fails or if there are errors for some of the Nodes, then the StatusCodes of these Nodes are mapped to the error code by the COM UA Proxy.

The mapping of the OPC UA Status code to OPC DA Read Error code (in the COM UA Proxy) is shown in Table A.8:

Table A.8 – OPC UA Read error mapping

OPC UA Status Code	OPC DA Error ID
Bad_OutOfMemory	E_OUTOFMEMORY
Bad_NodeIdInvalid	E_INVALIDITEMID
Bad_NodeIdUnknown	E_UNKNOWNITEMID
Bad_NotReadable	E_BADRIGHTS
Bad_UserAccessDenied	E_ACCESSDENIED
Bad_AttributeIdInvalid	E_INVALIDITEMID
Bad_UnexpectedError	E_FAIL
Bad_InternalError	E_FAIL
Bad_SessionClosed	E_FAIL
Bad_TypeMismatch	E_BADTYPE

A.4.5 Write data

The COM UA Proxy converts all the ItemIds in the Write into valid NodeIds by replacing the '-' with '='. It converts the Value, Quality and Timestamp (VQT) to a DataValue structure as per the mapping in Figure A.5, and calls the OPC UA Write Service for the Value Attribute.

If the Write Service call fails or if there are errors for some of the Nodes, then the StatusCodes of these Nodes are mapped to the error code by the COM UA Proxy.

The mapping of the OPC UA Status code to OPC DA Write Error code (in the COM UA Proxy) is shown in Table A.9.

Table A.9 – OPC UA Write error code mapping

OPC UA Status Code	OPC DA Error ID
Bad_TypeMismatch	E_BADTYPE
Bad_OutOfMemory	E_OUTOFMEMORY
Bad_NodeIdInvalid	E_INVALIDITEMID
Bad_NodeIdUnknown	E_UNKNOWNITEMID
Bad_NotWritable	E_BADRIGHTS
Bad_UserAccessDenied	E_ACCESSDENIED
Bad_AttributeIdInvalid	E_UNKNOWNITEMID
Bad_WriteNotSupported	E_NOTSUPPORTED
Bad_OutOfRange	E_RANGE

A.4.6 Subscriptions

The COM UA Proxy creates a Subscription in the OPC UA Server when a Group is created. The Name, Active flag, UpdateRate parameters of the Group are used while creating the subscription.

The COM UA Proxy Creates Monitored Items in the OPC UA Server when items are added to the Group.

The following parameters and filters are used for creating the monitored items:

- The *ItemIds* are converted to valid NodeIds by replacing the '-' with '='.
- Data Change Filter is used for Items with EU type as "Analog":
  - Trigger = STATUS_VALUE_1
  - if DeadBand value is specified for the *Group*:
    - DeadbandType = Percent_2
    - DeadbandValue = deadband specified for the group.

The COM UA Proxy calls the Publish Service of the OPC UA Server periodically and sends any data changes to the client.

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

ARCHITECTURE UNIFIÉE OPC –

Partie 8: Accès aux données

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 et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-8 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.

Cette troisième édition annule et remplace la deuxième édition parue en 2015. Cette édition constitue une révision technique.

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

a) ajout de nouveaux VariableTypes pour les AnalogItems;
b) ajout d'une annexe qui spécifie le mapping recommandé entre OPC UA DataAccess et OPC COM DataAccess;
c) modification de la description ambiguë de "Bad_NotConnected";

d) mise à jour de la description de EUInformation pour renvoyer à la dernière révision des unités CEFACT-ONU.

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

FDIS	Rapport de vote
65E/708/FDIS	65E/726/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette Norme internationale.

Ce document a été rédigé selon les Directives ISO/IEC, Partie 2.

Dans l'ensemble du présent document et dans les autres parties de la série IEC 62541, certaines conventions de document sont utilisées:

Le format *italique* est utilisé pour mettre en évidence un terme défini ou une définition qui apparaît à l'article "Termes et définitions" dans l'une des parties de la série IEC 62541.

Le format *italique* est également utilisé pour mettre en évidence le nom d'un paramètre d'entrée ou de sortie de service, ou le nom d'une structure ou d'un élément de structure habituellement défini dans les tableaux.

Par ailleurs, les *termes* et les *noms en italique* sont, à quelques exceptions près, écrits en camel-case (pratique qui consiste à joindre, sans espace, les éléments des mots ou expressions composés, la première lettre de chaque élément étant en majuscule). Par exemple, le terme défini est *AddressSpace* et non Espace d'adressage. Cela permet de mieux comprendre qu'il existe une définition unique pour *AddressSpace*, et non deux définitions distinctes pour Espace et pour Adressage.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Architecture unifiée OPC*, peut être consultée 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 "http://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 cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

ARCHITECTURE UNIFIÉE OPC –

Partie 8: Accès aux données

1 Domaine d'application

La présente partie de l'IEC 62541 fait partie intégrante de la série de normes générales sur l'architecture unifiée OPC (OPC UA). Elle définit le modèle d'information associé à l'Accès aux données (DA). Elle spécifie notamment des *VariableTypes* supplémentaires et fournit des descriptions complémentaires concernant les *NodeClasses* et *attributs* nécessaires pour l'Accès aux données, ainsi que des *propriétés* supplémentaires et d'autres paramètres relatifs aux informations et au comportement.

Le modèle d'espace d'adresses complet, comprenant toutes les *NodeClasses* et tous les *Attributs,* est spécifié dans l'IEC 62541-3. Les services de détection et d'accès aux données sont spécifiés dans l'IEC 62541-4.

2 Références normatives

Les documents ci-après sont des références normatives indispensables à l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1*, OPC Unified Architecture – Part 1: Overview and Concepts* (disponible en anglais seulement)

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*

CEFACT-ONU: Recommandation n° 20 de la CEE-ONU, *Codes for Units of Measure Used in International Trade*, disponible à l'adresse
https://www.unece.org/cefact/codesfortrade/codes_index.html

3 Termes, définitions et termes abrégés

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1, l'IEC 62541-3 et l'IEC 62541-4 ainsi que les suivants 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 http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1.1
DataItem
liaison à des données arbitraires et réelles d'automatisation, c'est-à-dire des données qui représentent des informations en cours de validité

Note 1 à l'article: Exemples de données:

- données relatives aux appareils (tels que des capteurs de température);
- données calculées;
- informations d'état (ouvert/fermé, en déplacement);
- données du système à variation dynamique (telles que les cours en Bourse);
- données de diagnostic.

3.1.2
AnalogItem
DataItem qui représente des grandeurs physiques continuellement variables (par exemple, longueur, température), par opposition à la représentation numérique des données des éléments discrets

Note 1 à l'article: Des exemples types sont les valeurs fournies par des capteurs de température ou de pression. L'OPC UA définit un *VariableType* particulier permettant d'identifier un *AnalogItem*. Les *propriétés* décrivent les plages possibles des *AnalogItems*.

3.1.3
DiscreteItem
DataItem qui représente des données qui ne peuvent avoir qu'un certain nombre de valeurs possibles (par exemple, OPENING, OPEN, CLOSING, CLOSED)

Note 1 à l'article: Des *VariableTypes* particuliers sont utilisés pour identifier les *DiscreteItems* à deux états ou plus. Les *propriétés* spécifient les valeurs de chaîne de ces états.

3.1.4
ArrayItem
DataItem qui représente des grandeurs physiques continuellement variables et où chaque point de données individuel comprend plusieurs valeurs représentées par une matrice (par exemple, la réponse spectrale d'un filtre numérique)

Note 1 à l'article: Des exemples types sont les données fournies par les appareils d'analyse. Des *VariableTypes* particuliers sont utilisés pour identifier les variantes d'*ArrayItems*.

3.1.5
EngineeringUnits
unités de mesure des *AnalogItems* qui représentent des grandeurs physiques continuellement variables (par exemple, longueur, masse, temps, température)

Note 1 à l'article: La présente norme définit des *propriétés* qui indiquent l'unité utilisée pour la valeur du *DataItem*, ainsi que la valeur la plus élevée et la plus basse susceptibles d'être rencontrées en fonctionnement normal.

3.2 Termes abrégés

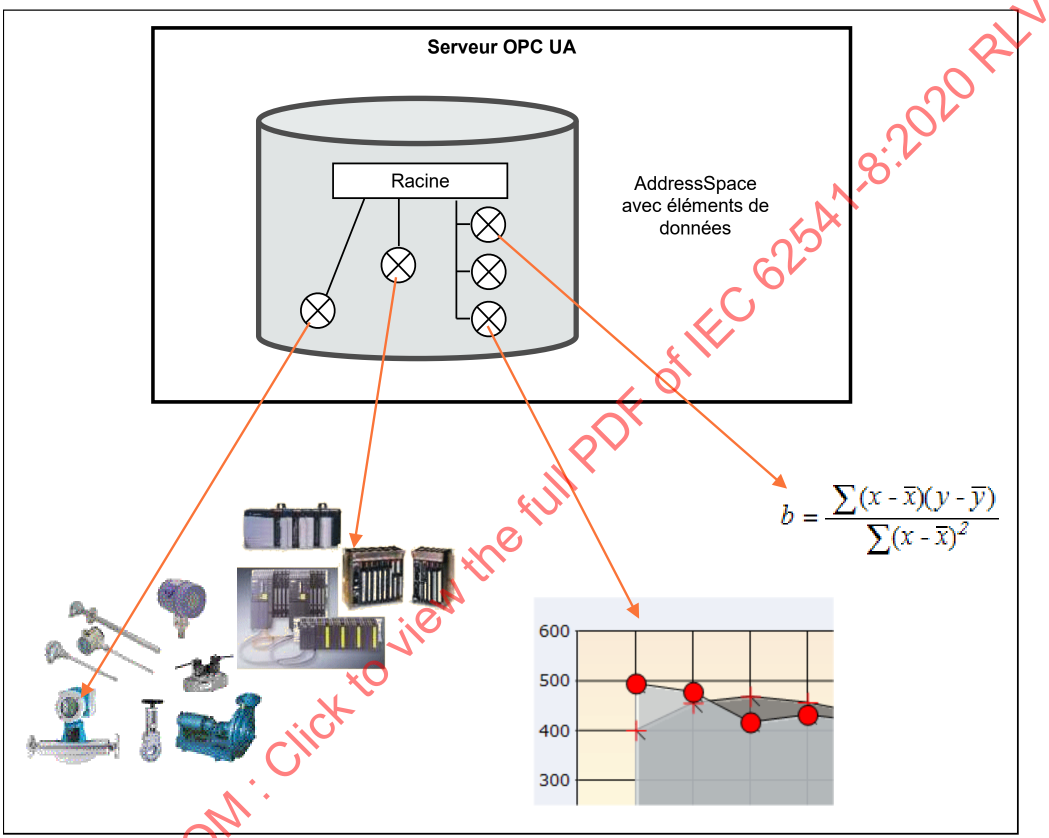
DA	data access (accès aux données)
EU	engineering unit (unité technique)
UA	Unified Architecture (Architecture unifiée)

4 Concepts

L'Accès aux données désigne la représentation et l'utilisation des données d'automatisation dans les Serveurs.

Les données d'automatisation peuvent résider dans le *Serveur* ou sur des cartes E/S directement connectées au *Serveur*. Elles peuvent également résider dans des sous-serveurs ou sur d'autres appareils tels que des contrôleurs et des modules d'entrée/sortie, connectés par des liaisons série par l'intermédiaire de bus de terrain ou d'autres liaisons de communication. Les *Serveurs* OPC UA d'accès aux données fournissent aux *Clients* OPC UA un ou plusieurs Accès aux données en offrant un accès transparent à leurs données d'automatisation.

Les liaisons aux instances de données d'automatisation sont appelées *DataItems*. Les catégories de données d'automatisation fournies sont spécifiques au fournisseur. La Figure 1 montre comment l'*AddressSpace* d'un *Serveur* peut comporter une vaste plage de *DataItems*.


Serveur OPC UA
Racine
AddressSpace avec éléments de données
$b = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$
600
500
400
300

IEC

Figure 1 – *DataItems* OPC reliés aux données d'automatisation

Les *Clients* peuvent lire ou écrire les *DataItems* ou surveiller la modification des valeurs. Les *Services* nécessaires à ces opérations sont spécifiés dans l'IEC 62541-4. Les modifications sont définies comme une modification d'état (qualité) ou une modification de valeur qui dépasse une plage définie par le client, appelée *Deadband*. Afin de détecter la modification de valeur, la différence entre la valeur actuelle et la dernière valeur récupérée est comparée à la *Deadband*.

5 Modèle

5.1 Généralités

Le modèle DataAccess étend le modèle de variable en définissant des *VariableTypes*. Le *DataItemType* est le type de base. *ArrayItemType*, *BaseAnalogType* et *DiscreteItemType* sont des spécialisations. Voir Figure 2. Chacun de ces *VariableTypes* peut être étendu pour former des *DataItems* spécifiques à un domaine ou à un serveur.

L'Annexe A spécifie la manière recommandée pour mapper les informations provenant des serveurs OPC COM DA au modèle de ce document.

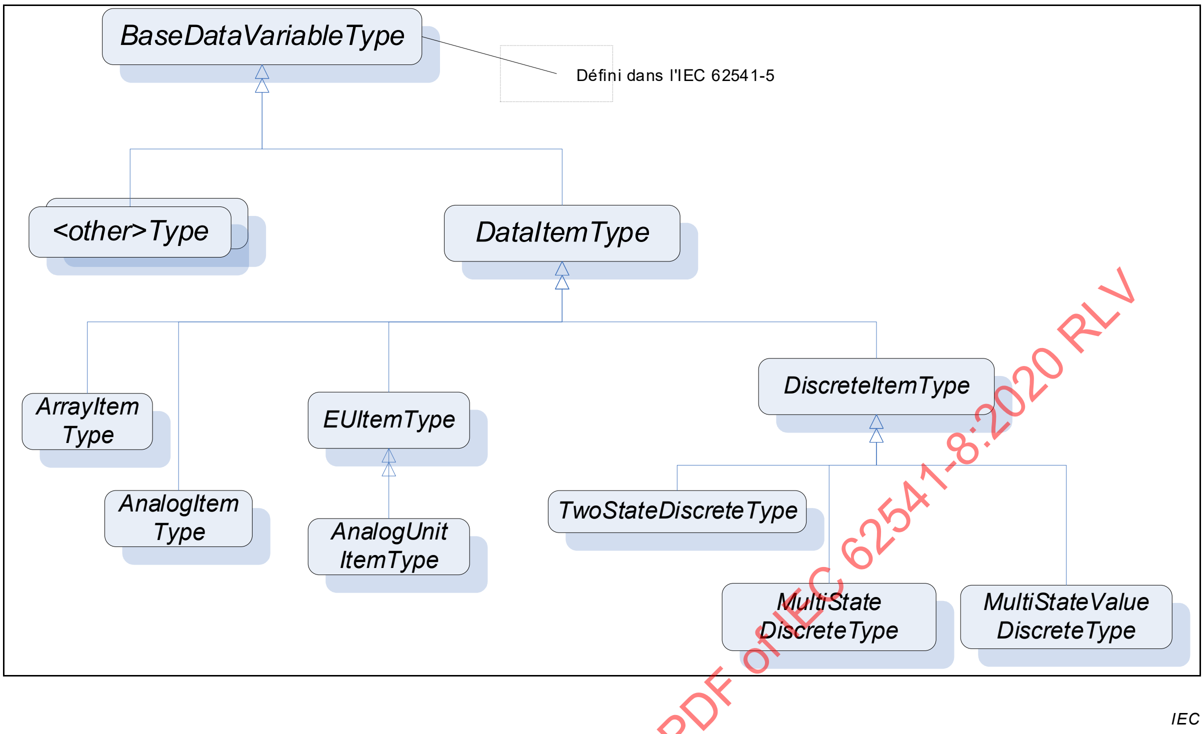


Figure 2 – Hiérarchie du *VariableType DataItem*

5.2 SemanticsChanged

Le *StatusCode* contient également un bit d'information appelé *SemanticsChanged*.

Les *Serveurs* qui mettent en œuvre l'Accès aux données doivent définir ce Bit dans les notifications en cas de modification de certaines *propriétés* définies dans la présente norme. Les *propriétés* correspondantes sont spécifiées de manière individuelle pour chaque *VariableType*.

Il convient que les *Clients* qui utilisent l'une de ces *propriétés* les relisent avant de traiter la valeur de données.

5.3 Types de variables

5.3.1 DataItemType

Ce *VariableType* définit les caractéristiques générales d'un *DataItem*. Tous les autres types de *DataItems* sont issus de ce type de variable. Le *DataItemType* est dérivé du *BaseDataVariableType* et partage par conséquent le modèle de variable, comme décrit dans l'IEC 62541-3 et l'IEC 62541-5. Il est défini de manière formelle dans le Tableau 1.

Tableau 1 – Définition de DataItemType

Attribut	Valeur				
BrowseName	DataItemType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseDataVariableType* défini dans l'IEC 62541-5; c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasSubtype	VariableType	AnalogItemType	Défini en 5.3.2		
HasSubtype	VariableType	DiscreteItemType	Défini en 5.3.3		
HasSubtype	VariableType	ArrayItemType	Défini en 5.3.4		
HasProperty	Variable	Definition	String	PropertyType	Optional
HasProperty	Variable	ValuePrecision	Double	PropertyType	Optional

La *propriété Definition* est une chaîne en clair et spécifique au fournisseur qui définit le mode de calcul de la valeur de ce *DataItem*. La *propriété Definition* n'est pas localisée et contient le plus souvent une équation qui peut être analysée par certains clients.

EXEMPLE: *Definition*::= "(TempA – 25) + TempB"

La *propriété ValuePrecision* spécifie la précision maximale que le *Serveur* peut procurer pour l'élément en fonction des restrictions qui existent dans l'environnement cible.

La *propriété ValuePrecision* peut être utilisée pour les *DataTypes* suivants:

- pour les valeurs de "Float" et "Double", elle spécifie le nombre de chiffres après la virgule;
- pour les valeurs de DateTime, elle indique l'intervalle minimal en nanosecondes. Par exemple, une *ValuePrecision* de 20 000 000 définit une précision de 20 ms.

La *propriété ValuePrecision* est une approximation dont l'objet est de fournir une recommandation pour le *Client*. Le *Serveur* est supposé arrondir les valeurs discrètement avec plus de précision. Autrement dit, un *Client* peut être confronté à une situation où la valeur relue et renvoyée par le *Serveur* est différente de celle qu'il a écrite et envoyée au *Serveur*. Cette différence ne doit pas être supérieure à celle spécifiée par cette *Propriété*.

5.3.2 VariableTypes AnalogItem

5.3.2.1 Généralités

Les *VariableTypes* définies dans le présent paragraphe définissent les caractéristiques des *AnalogItems*. Ces types ont une sémantique et des *propriétés* identiques, mais possèdent des *ModellingRules* différentes pour les *propriétés* individuelles.

Les *propriétés* sont décrites en 5.3.2.2. Les descriptions s'appliquent aux *Propriétés* pour les autres *VariableTypes* également.

5.3.2.2 BaseAnalogType

Ce *VariableType* est le type de base des éléments analogiques. Toutes les *propriétés* sont facultatives. Les sous-types de ce type de base exigent certaines *propriétés*. Le *BaseAnalogType* est dérivé du *DataItemType*. Il est défini de manière formelle dans le Tableau 2.

Tableau 2 – Définition de BaseAnalogType

Attribut	**Valeur**				
BrowseName	BaseAnalogType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DataItemType* défini en 5.3.1, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasSubtype	VariableType	AnalogItemType	Défini en 5.3.2.3.		
HasSubtype	VariableType	AnalogUnitType	Défini en 5.3.2.4.		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Optional
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional

Les alinéas suivants décrivent les *propriétés* de ce *VariableType*. Si la *valeur* de l'élément analogique contient une matrice, les *propriétés* doivent s'appliquer à tous les éléments de la matrice.

La *propriété InstrumentRange* définit la plage de valeurs que l'instrument peut renvoyer.

EXEMPLE 1 *InstrumentRange*::= {−9 999,9, 9 999,9}

Bien qu'elle soit définie comme facultative, il est fortement recommandé que les *Serveurs* prennent en charge cette *Propriété*. En l'absence d'*InstrumentRange*, les *Clients* utilisent généralement la plage complète selon le *DataType*.

La *propriété InstrumentRange* peut également être utilisée pour restreindre un *DataType* intégré (comme Byte ou Int16) à une plage de valeurs plus étroites.

EXEMPLE 2

Uint4:	*InstrumentRange*::= {0, 15}
Int6:	*InstrumentRange*::= {-32, 31}

Le *DataType Range* est spécifié en 5.6.2.

EURange définit la plage de valeurs susceptibles d'être rencontrées en fonctionnement normal. Elle est destinée à être utilisée comme mise à l'échelle automatique d'un affichage de diagramme à barres.

La défaillance ou la désactivation d'un capteur ou d'un instrument peut donner lieu au renvoi d'une valeur d'élément qui est en réalité en dehors de cette plage. Il importe que le logiciel *Client* soit préparé à gérer cette possibilité. De même, un *Client* peut tenter d'écrire au serveur une valeur qui est en dehors de cette plage. Dans ce cas, le comportement exact (accepter, refuser, retenir, etc.) dépend du *Serveur*. Cependant, les *Serveurs* doivent généralement être préparés à gérer cette situation.

EXEMPLE 3 *EURange*::= {−200,0, 1 400,0}

Pour plus d'informations sur le filtre de surveillance spécial (*PercentDeadband*) qui repose sur la plage d'unités techniques, voir aussi 6.2.

NOTE 1 Si *EURange* n'est pas indiqué sur une instance, le filtre *PercentDeadband* ne peut pas être utilisé pour cette instance (voir 6.2).

Les *EngineeringUnits* spécifient les unités de la valeur de *DataItem* (par exemple, DEGC, hertz, secondes). Le type *EUInformation* est spécifié en 5.6.3.

Il est important de noter l'importance qu'il est essentiel de comprendre les unités d'une valeur de mesure dans le cadre d'un système uniforme. Dans un système ouvert, notamment lorsque des *Serveurs* issus de différents environnements peuvent être utilisés, il est primordial de connaître les unités de mesure. A partir de ces connaissances, les valeurs peuvent être converties si nécessaire avant leur utilisation. Par conséquent, bien qu'elle soit définie comme facultative, la prise en charge de la *propriété EngineeringUnits* est fortement recommandée.

L'OPC UA recommande d'utiliser les "codes des unités de mesure" (voir CEFACT-ONU: Recommandation n° 20 de la CEE-ONU). La correspondance avec la *propriété EngineeringUnits* est spécifiée en 5.6.3.

NOTE 2 Exemple de confusion d'unités: en 1999, la sonde Mars Climate Orbiter s'est écrasée sur la surface de la planète Mars. La cause principale de l'incident était une discordance concernant les unités utilisées. Le logiciel de navigation attendait des données en newton-seconde, tandis que la société qui avait construit le véhicule orbital avait fourni des données en livres-force par seconde. Le même problème (certes moins coûteux) se produit lorsque les clients habitués aux pintes britanniques commandent une pinte aux Etats-Unis et qu'on leur sert simplement ce qu'ils jugent comme une "tromperie sur la marchandise".

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés EURange* (ce qui peut modifier le comportement d'un *Abonnement* s'il utilise un filtre à *PercentDeadband*) ou *EngineeringUnits* (ce qui peut engendrer des problèmes si le *Client* utilise la valeur pour réaliser des calculs) (voir 5.2 pour plus d'informations).

5.3.2.3 AnalogItemType

Ce *VariableType* exige la *propriété EURange*. L'*AnalogItemType* est dérivé du *BaseAnalogType*. Il est défini de manière formelle dans le Tableau 3.

Tableau 3 – Définition d'*AnalogItemType*

Attribut	**Valeur**				
BrowseName	AnalogItemType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseAnalogType* défini en 5.3.2.2, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasSubtype	VariableType	AnalogUnitRangeType	Défini en 5.3.2.5		
HasProperty	Variable	EURange	Range	PropertyType	Mandatory

5.3.2.4 AnalogUnitType

Ce *VariableType* exige la *propriété EngineeringUnits*. L'*AnalogUnitType* est dérivé du *BaseAnalogType*. Il est défini de manière formelle dans le Tableau 4.

Tableau 4 – Définition d'AnalogUnitType

Attribut	**Valeur**				
BrowseName	AnalogUnitType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *BaseAnalogType* défini en 5.3.2.2, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.2.5 AnalogUnitRangeType

L'*AnalogUnitRangeType* est dérivé d'*AnalogItemType* et exige en outre la *propriété EngineeringUnitsType*. Il est défini de manière formelle dans le Tableau 5.

Tableau 5 – Définition d'AnalogUnitRangeType

Attribut	**Valeur**				
BrowseName	AnalogUnitRangeType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Number				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*AnalogtemType* défini en 5.3.2.3, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory

5.3.3 DiscreteItemType

5.3.3.1 Généralités

Ce VariableType est un type abstrait. En d'autres termes, aucune instance de ce type ne peut exister. Cependant, il peut être utilisé dans un filtre lors de la navigation ou de l'interrogation. Le *DiscreteItemType* est dérivé du *DataItemType* et partage par conséquent toutes ses caractéristiques. Il est défini de manière formelle dans le Tableau 6.

Tableau 6 – Définition de DiscreteItemType

Attribut	**Valeur**				
BrowseName	DiscreteItemType				
IsAbstract	True				
ValueRank	−2 (−2 = 'Any')				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DataItemType* défini en 5.2, c'est-à-dire que les *propriétés* de ce type sont héritées					
HasSubtype	VariableType	TwoStateDiscreteType	Défini en 5.3.3.2		
HasSubtype	VariableType	MultiStateDiscreteType	Défini en 5.3.3.3		
HasSubtype	VariableType	MultiStateValueDiscreteType	Défini en 5.3.3.4		

5.3.3.2 TwoStateDiscreteType

Ce *VariableType* définit les caractéristiques générales d'un *DiscreteItem* qui peut avoir deux états. Le *TwoStateDiscreteType* est dérivé du *DiscreteItemType*. Il est défini de manière formelle dans le Tableau 7.

Tableau 7 – Définition de TwoStateDiscreteType

Attribut	Valeur				
BrowseName	TwoStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	Boolean				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DiscreteItemType* défini en 5.3.3, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasProperty	Variable	TrueState	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	FalseState	LocalizedText	PropertyType	Mandatory

TrueState contient une chaîne à associer à ce *DataItem* lorsqu'il est TRUE. Il est généralement utilisé pour un contact lorsqu'il est à l'état fermé (non nul).

Par exemple: "RUN", "CLOSE", "ENABLE", "SAFE", etc.

FalseState contient une chaîne à associer à ce *DataItem* lorsqu'il est FALSE. Il est généralement utilisé pour un contact lorsqu'il est à l'état ouvert (zéro).

Par exemple: "STOP", "OPEN", "DISABLE", "UNSAFE", etc.

Si l'élément contient une matrice, les *propriétés* s'appliquent alors à tous les éléments de la matrice.

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés FalseState ou TrueState* (des modifications peuvent entraîner une mauvaise interprétation de la part les utilisateurs ou les programmes (de script)) (voir 5.2 pour plus d'informations).

5.3.3.3 MultiStateDiscreteType

Ce *VariableType* définit les caractéristiques générales d'un *DiscreteItem* qui peut avoir plus de deux états. Le *MultiStateDiscreteType* est dérivé du *DiscreteItemType*. Il est défini de manière formelle dans le Tableau 8.

Tableau 8 – Définition de MultiStateDiscreteType

Attribut	Valeur				
BrowseName	MultiStateDiscreteType				
IsAbstract	False				
ValueRank	−2 (−2 = 'Any')				
DataType	UInteger				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DiscreteItemType* défini en 5.3.3, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasProperty	Variable	EnumStrings	LocalizedText[]	PropertyType	Mandatory

EnumStrings est une table de consultation de chaîne correspondant à des valeurs numériques séquentielles (0, 1, 2, etc.)

Exemples:

"OPEN"
"CLOSE"
"IN TRANSIT", etc.

Dans ce cas, la chaîne "OPEN" correspond à 0, "CLOSE" à 1 et "IN TRANSIT" à 2.

Il convient que les clients soient préparés à gérer les valeurs d'éléments en dehors de la plage de la liste. Il convient que des serveurs robustes soient préparés à gérer les écritures de valeurs non valides.

Si l'élément contient une matrice, cette table de consultation doit alors s'appliquer à tous les éléments de la matrice.

NOTE La propriété *EnumStrings* est également utilisée pour les *DataTypes* de type Enumeration (pour la spécification de ce *DataType*, voir IEC 62541-3).

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de la *propriété EnumStrings* (des modifications peuvent entraîner une mauvaise interprétation par les utilisateurs ou les programmes (de script)) (voir 5.2 pour plus d'informations).

5.3.3.4 MultiStateValueDiscreteType

Ce *VariableType* définit les caractéristiques générales d'un *DiscreteItem* qui peut avoir plus de deux états et dont les valeurs d'état (l'énumération) ne consistent pas en des valeurs numériques consécutives (il peut y avoir des espaces) ou avec lesquels l'énumération ne repose pas sur zéro. Le *MultiStateValueDiscreteType* est dérivé du *DiscreteItemType*. Il est défini de manière formelle dans le Tableau 9.

Tableau 9 – Définition de MultiStateValueDiscreteType

Attribut	**Valeur**				
BrowseName	MultiStateValueDiscreteType				
IsAbstract	False				
ValueRank	Scalar				
DataType	Number				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DiscreteItemType* défini en 5.3.3, c'est-à-dire que les *propriétés* de ce type sont héritées.					
HasProperty	Variable	EnumValues	Voir IEC 62541-3		Mandatory
HasProperty	Variable	ValueAsText	Voir IEC 62541-3		Mandatory

Les *EnumValues* constituent une matrice d'*EnumValueType*. Chaque entrée de la matrice comporte une valeur d'énumération avec sa notation sous forme d'entiers, une représentation en clair et des informations d'aide. Ceci représente des énumérations avec des entiers qui ne sont pas à base zéro ou comportent des espaces (par exemple, 1, 2, 4, 8, 16). Voir l'IEC 62541-3 pour la définition de ce type. Les *variables MultiStateValueDiscrete* présentent la notation actuelle sous forme d'entiers dans leur *attribut Value*. Les *Clients* lisent souvent la *Ppropriété EnumValues* à l'avance et la mettent en cache afin de rechercher un nom ou de l'aide chaque fois qu'ils reçoivent la représentation numérique.

Seuls les *DataTypes* qui peuvent être représentés avec des *EnumValues* sont admis pour les *variables* de *MultiStateValueDiscreteType*. Ceux-ci sont les suivants:

- entiers non signés jusqu'à une longueur de 64 bits;
- entiers non signés jusqu'à une longueur de 63 bits.

La représentation numérique de la valeur d'énumération actuelle est fournie via l'*attribut Value* de la *variable MultiStateValueDiscrete*. La *propriété ValueAsText* fournit la représentation en texte localisé de la valeur d'énumération. Elle peut être utilisée par les *Clients* qui s'intéressent uniquement à l'affichage du texte d'abonnement à la *propriété* en lieu et place de l'*attribut Value*.

5.3.4 ArrayItemType

5.3.4.1 Généralités

Ce *VariableType* abstrait définit les caractéristiques générales d'un *ArrayItem*. Les valeurs sont présentées dans une matrice, mais le contenu de cette matrice représente une entité unique telle qu'une image. Les autres *DataItems* peuvent contenir des matrices qui représentent, par exemple, plusieurs valeurs de différents capteurs de température d'une chaudière.

L'*ArrayItemType* ou son sous-type doit être utilisé uniquement lorsque les *propriétés Title* et *AxisScaleType* peuvent être remplies avec des valeurs justifiées. Si cela n'est pas le cas, le *DataItemType* et les sous-types tels que l'*AnalogItemType* prenant également en charge les matrices doivent être utilisés. *ArrayItemType* est défini de manière formelle dans le Tableau 10.

Tableau 10 – Définition d'ArrayItemType

Attribut	Valeur				
BrowseName	ArrayItemType				
IsAbstract	True				
ValueRank	0 (0 = OneOrMoreDimensions)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type du *DataItemType* défini en 5.3.1, c'est-à-dire que les *propriétés* de ce type sont héritées					
HasSubtype	VariableType	YArrayItemType	Défini en 5.3.4.2		
HasSubtype	VariableType	XYArrayItemType	Défini en 5.3.4.3		
HasSubtype	VariableType	ImageItemType	Défini en 5.3.4.4		
HasSubtype	VariableType	CubeItemType	Défini en 5.3.4.5		
HasSubtype	VariableType	NDimensionArrayItemType	Défini en 5.3.4.6		
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Mandatory
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Mandatory
HasProperty	Variable	Title	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	AxisScaleType	AxisScaleEnumeration	PropertyType	Mandatory

InstrumentRange définit la plage de la *Valeur* de l'*ArrayItem.*

EURange définit la plage de valeurs de l'*ArrayItem* susceptibles d'être rencontrées en fonctionnement normal. Elle est destinée à être utilisée comme mise à l'échelle automatique d'un affichage de diagramme à barres.

EngineeringUnits comporte des informations sur les unités techniques de la *Valeur* de l'*ArrayItem*.

Pour les informations supplémentaires concernant *InstrumentRange*, *EURange* et *EngineeringUnits*, voir la description de l'*AnalogItemType* en 5.3.2.

La propriété *Title* comporte l'intitulé lisible par l'utilisateur de la *valeur* de l'*ArrayItem*.

AxisScaleType définit l'échelle à utiliser pour l'axe où la *Valeur* de l'*ArrayItem* doit être affichée.

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés InstrumentRange, EURange, EngineeringUnits* ou *Title* (voir 5.2 pour plus d'informations).

5.3.4.2 YArrayItemType

Le *YArrayItemType* représente une matrice unidimensionnelle de valeurs numériques utilisée pour représenter des spectres ou des distributions, où les intervalles de l'axe X (abscisses) sont constants. *YArrayItemType* est défini de manière formelle dans le Tableau 11.

Tableau 11 – Définition de YArrayItemType

Attribut	Valeur				
BrowseName	YArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	BaseDataType				
ArrayDimensions	{0} (0 = UnknownSize)				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*ArrayItemType* défini en 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

La *valeur* de *YArrayItem* contient les valeurs numériques pour l'axe Y. Les *unités techniques* et la *plage* applicable à la *valeur* sont définies par les *propriétés* correspondantes héritées de l'*ArrayItemType*.

Le *DataType* de ce *VariableType* est limité à SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* et *DoubleComplexNumberType*.

La *propriété XAxisDefinition* contient des informations sur les *Unités techniques* et la *Plage* applicables à l'axe X.

Le bit de StatusCode SemanticsChanged doit être défini en cas de modification de l'une des cinq *propriétés* suivantes: *InstrumentRange, EURange, EngineeringUnits, Title* ou *XAxisDefinition* (voir 5.2 pour plus d'informations).

La Figure 3 représente un exemple de la manière dont les *attributs* et les *propriétés* peuvent être utilisés dans une interface graphique.

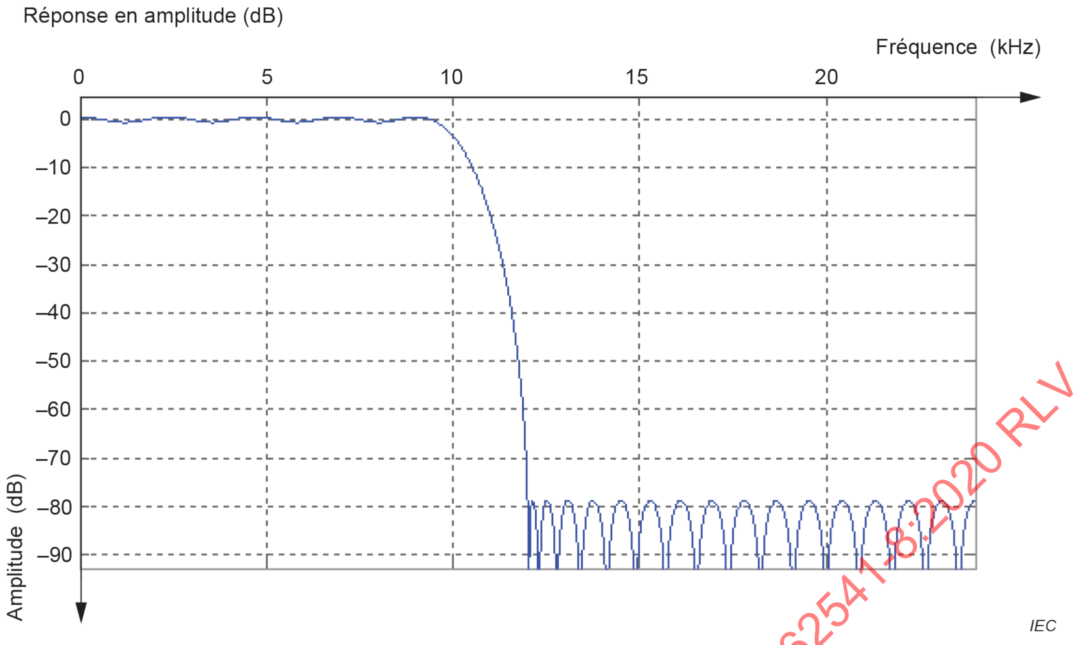


Figure 3 – Représentation graphique d'un *YArrayItem*

Le Tableau 12 décrit les valeurs de chaque élément représenté à la Figure 3.

Tableau 12 – Description de l'élément *YArrayItem*

Attribut/Propriété	Valeur d'élément
Description	Réponse en amplitude (dB)
axisScaleType	AxisScaleEnumeration.LINEAR_0
InstrumentRange.low	-90
InstrumentRange.high	5
EURange.low	-90
EURange.high	2
EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
EngineeringUnits.unitId	2N
EngineeringUnits.displayName	"en-us", "dB"
EngineeringUnits.description	"en-us", "decibel"
Title	Amplitude
XAxisDefinition.EngineeringUnits.namespaceUrl	http://www.opcfoundation.org/UA/units/un/cefact
XAxisDefinition.EngineeringUnits.unitId	kHz
XAxisDefinition.EngineeringUnits.displayName	"en-us", "kHz"
XAxisDefinition.EngineeringUnits.description	"en-us", "kilohertz"
XAxisDefinition.Range.low	0
XAxisDefinition.Range.high	25
XAxisDefinition.title	"en-us", "Frequency"
XAxisDefinition.axisScaleType	AxisScaleEnumeration.LINEAR_0
XAxisDefinition.axisSteps	null
Notes d'interprétation: • Les éléments de ce tableau ne sont pas tous utilisés dans la Figure 3. • L'axe X est affiché selon un ordre inversé. Cependant, *XAxisDefinition.Range.low* doit être inférieure à *XAxisDefinition.Range.high*. Seule une représentation graphique inverse l'ordre d'affichage. • L'axe X est constant.	

5.3.4.3 XYArrayItemType

L'*XYArrayItemType* représente un vecteur des valeurs de XVType telles qu'une liste des valeurs de crête, où XVType.x et XVType.value sont respectivement la position et l'intensité de la valeur crête. *XYArrayItemType* est défini de manière formelle dans le Tableau 13.

Tableau 13 – Définition de XYArrayItemType

Attribut	**Valeur**				
BrowseName	XYArrayItemType				
IsAbstract	False				
ValueRank	1				
DataType	XVType (défini en 5.6.8)				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*ArrayItemType* défini en 5.3.4.1					
HasProperty	Variable	XAxisDefinition	AxisInformation	PropertyType	Mandatory

La *valeur* de *XYArrayItem* contient une matrice de structures (XVType), où chaque structure spécifie la position de l'axe X (XVType.x) et la valeur proprement dite (XVType.value) utilisée pour l'axe Y. Les unités techniques et la plage applicable à la *valeur* sont définies par les *propriétés* correspondantes héritées de l'*ArrayItemType*.

La *propriété XAxisDefinition* contient des informations sur les *Unités techniques* et la *Plage* applicables à l'axe X.

Les *axisSteps* de la *XAxisDefinition* doivent être mis à NULL, car ils ne sont pas utilisés.

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés InstrumentRange, EURange, EngineeringUnits, Title* ou *XAxisDefinition* (voir 5.2 pour plus d'informations).

5.3.4.4 ImageItemType

L'*ImageItemType* définit les caractéristiques générales d'un ImageItem qui représente une matrice de valeurs telle qu'une image, où la position des pixels est donnée par X et Y qui correspondent respectivement à la colonne et à la rangée. La valeur correspond à l'intensité des pixels.

ImageItemType est défini de manière formelle dans le Tableau 14.

Tableau 14 – Définition d'ImageItemType

Attribut	**Valeur**				
BrowseName	ImageItemType				
IsAbstract	False				
ValueRank	2 (2 = two dimensional array)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*ArrayItemType* défini en 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Les unités techniques et la plage applicable à la *valeur* sont définies par les *propriétés* correspondantes héritées de l'*ArrayItemType*.

Le *DataType* de ce *VariableType* est limité à SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* et *DoubleComplexNumberType*.

L'*attribut ArrayDimensions* pour les *variables* de ce type ou des sous-types doit utiliser la première entrée ([0]) dans la matrice pour définir le nombre de colonnes et la seconde entrée ([1]) pour définir le nombre de rangées en retenant l'hypothèse que la taille de la matrice n'est pas dynamique.

La *propriété XAxisDefinition* contient des informations sur les unités techniques et la plage applicables à l'axe X.

La *propriété YAxisDefinition* contient des informations sur les unités techniques et la plage applicables à l'axe Y.

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition* ou *YAxisDefinition*.

5.3.4.5 CubeItemType

Le *CubeItemType* représente un cube de valeurs tel qu'une distribution de particules dans l'espace, où la position des particules est donnée par X, Y et Z qui correspondent respectivement à la colonne, la rangée et la profondeur. Dans l'exemple d'une distribution des particules dans l'espace, la valeur correspond à la granulométrie. *CubeItemType* est défini de manière formelle dans le Tableau 15.

Tableau 15 – Définition de CubeItemType

Attribut	**Valeur**				
BrowseName	CubeItemType				
IsAbstract	False				
ValueRank	3 (3 = three dimensional array)				
DataType	BaseDataType				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type de l'*ArrayItemType* défini en 5.3.4.1					
HasProperty	Variable	XAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	YAxisDefinition	*AxisInformation*	PropertyType	Mandatory
HasProperty	Variable	ZAxisDefinition	*AxisInformation*	PropertyType	Mandatory

Les unités techniques et la plage applicable à la *valeur* sont définies par les *propriétés* correspondantes héritées de l'*ArrayItemType*.

Le *DataType* de ce *VariableType* est limité à SByte, Int16, Int32, Int64, Float, Double, *ComplexNumberType* et *DoubleComplexNumberType*.

Il convient que l'*attribut ArrayDimensions* pour les *variables* de ce type ou les sous-types utilise la première entrée ([0]) dans la matrice pour définir le nombre de colonnes, la deuxième entrée ([1]) pour définir le nombre de rangées et la troisième entrée ([2]) pour définir le nombre de paliers de l'axe Z en retenant l'hypothèse que la taille de la matrice n'est pas dynamique.

La *propriété XAxisDefinition* contient des informations sur les unités techniques et la plage applicables à l'axe X.

La *propriété YAxisDefinition* contient des informations sur les unités techniques et la plage applicables à l'axe Y.

La *propriété ZAxisDefinition* contient des informations sur les unités techniques et la plage applicables à l'axe Z.

Le bit de *StatusCode SemanticsChanged* doit être défini en cas de modification de l'une des *propriétés InstrumentRange, EURange, EngineeringUnits, Title, XAxisDefinition, YAxisDefinition* ou *ZAxisDefinition* (voir 5.2 pour plus d'informations).

5.3.4.6 NDimensionArrayItemType

Ce *VariableType* définit un *ArrayItem* à plusieurs dimensions.