

OPC Unified Architecture

Specification

Part 8: Data Access

Release 1.01

February 6, 2009

Specification Type:	Industry Standard Specification	Comments:	Report or view errata: http://www.opcfoundation.org/errata
Title:	OPC Unified Architecture Part 8 :Data Access	Date:	February 6, 2009
Version:	Release 1.01	Software: Source:	MS-Word OPC UA Part 8 - Data Access 1.01 Specification.doc
Author:	OPC Foundation	Status:	Release

CONTENTS

Page

FOREWORD	iv
<u>AGREEMENT OF USE</u>	iv
1 Scope	1
2 Reference documents	1
3 Terms, definitions, and abbreviations	1
3.1 OPC UA Part 1 terms	1
3.2 OPC UA Part 3 terms	1
3.3 OPC UA Part 4 terms	2
3.4 OPC UA Data Access terms	2
3.4.1 DataItem	2
3.4.2 AnalogItem	2
3.4.3 DiscreteItem	2
3.4.4 EngineeringUnits	2
3.5 Abbreviations and symbols	2
4 Concepts	3
5 Model	4
5.1 General	4
5.2 Variable Types	4
5.2.1 DataItemType	4
5.2.2 AnalogItemType	5
5.2.3 DiscreteItemType	6
5.3 Address Space Model	8
5.4 Attributes of DataItems	8
5.5 Property DataTypes	9
5.5.1 Overview	9
5.5.2 Range	9
5.5.3 EUInformation	10
6 Data Access specific usage of Services	11
6.1 PercentDeadband	11
6.2 Data Access Status Codes	11
6.2.1 Overview	11
6.2.2 Operation level result codes	11
6.2.3 LimitBits	12
6.2.4 SemanticsChanged	13

FIGURES

Figure 1 – OPC *DataItems* are linked to automation data 3

Figure 2 – *DataItem VariableType* Hierarchy 4

Figure 3 – Representation of *DataItems* in the *AddressSpace* 8

TABLES

Table 1 – <i>DataItemType</i> Definition	4
Table 2 – <i>AnalogItemType</i> Definition	5
Table 3 – <i>DiscreteItemType</i> Definition	6
Table 4 – <i>TwoStateDiscreteType</i> Definition	6
Table 5 – <i>MultiStateDiscreteType</i> Definition	7
Table 6 – <i>Range</i> Data Type Structure	9
Table 7 – <i>Range</i> Definition	9
Table 8 – <i>EUInformation</i> Data Type Structure	10
Table 9 – <i>EUInformation</i> Definition	10
Table 10 – <i>Bad</i> operation level result codes	12
Table 11 – <i>Uncertain</i> operation level result codes	12
Table 12 – <i>Good</i> operation level result codes	12

OPC FOUNDATION

UNIFIED ARCHITECTURE –

FOREWORD

This specification is the specification for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that shall inter-operate seamlessly together.

Copyright © 2006-2009, OPC Foundation, Inc.

AGREEMENT OF USE

COPYRIGHT RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

OPC Foundation members and non-members are prohibited from copying and redistributing this specification. All copies must be obtained on an individual basis, directly from the OPC Foundation Web site <http://www.opcfoundation.org>.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OPC specifications may require use of an invention covered by patent rights. OPC shall not be responsible for identifying patents for which a license may be required by any OPC specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OPC specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OPC FOUNDATION MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OPC FOUNDATION BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you.

RESTRICTED RIGHTS LEGEND

This Specification is provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation, 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The OPC Foundation shall at all times be the sole entity that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice of law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here: <http://www.opcfoundation.org/errata>

OPC Unified Architecture Specification

Part 8: Data Access

1 Scope

This specification is part of the overall OPC Unified Architecture (OPC UA) specification 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 Part 3. The services to detect and access data are specified in Part 4.

2 Reference documents

Part 1: OPC UA Specification: Part 1 – Concepts, Version 1.01 or later

<http://www.opcfoundation.org/UA/Part1/>

Part 3: OPC UA Specification: Part 3 – Address Space Model, Version 1.01 or later

<http://www.opcfoundation.org/UA/Part3/>

Part 4: OPC UA Specification: Part 4 – Services, Version 1.01 or later

<http://www.opcfoundation.org/UA/Part4/>

Part 5: OPC UA Specification: Part 5 – Information Model, Version 1.01 or later

<http://www.opcfoundation.org/UA/Part5/>

3 Terms, definitions, and abbreviations

3.1 OPC UA Part 1 terms

The following terms defined in Part 1 apply.

- AddressSpace
- Node
- NodeClass
- Reference
- Subscription
- View

3.2 OPC UA Part 3 terms

The following terms defined in Part 3 apply.

- DataVariable
- Object
- Property
- Variable
- VariableType

3.3 OPC UA Part 4 terms

The following terms defined in Part 4 apply.

- Deadband

3.4 OPC UA Data Access terms

3.4.1 DataItem

a link to arbitrary, live automation data, i.e. data that represents current information.

NOTE: 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

3.4.2 AnalogItem

DataItems that represent continuously-variable physical quantities.

NOTE: 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*.

3.4.3 DiscreteItem

DataItems that represent data that may take on only a certain number of possible values.

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

3.4.4 EngineeringUnits

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

NOTE: This Part 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.

3.5 Abbreviations 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 OPC UA Servers.

Automation data can be located inside the OPC UA Server or on I/O cards directly connected to the OPC UA 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*. Which categories of automation data are provided is completely vendor-specific. Figure 1 illustrates how the *AddressSpace* of an OPC UA server might consist of a broad range of different *DataItems*.

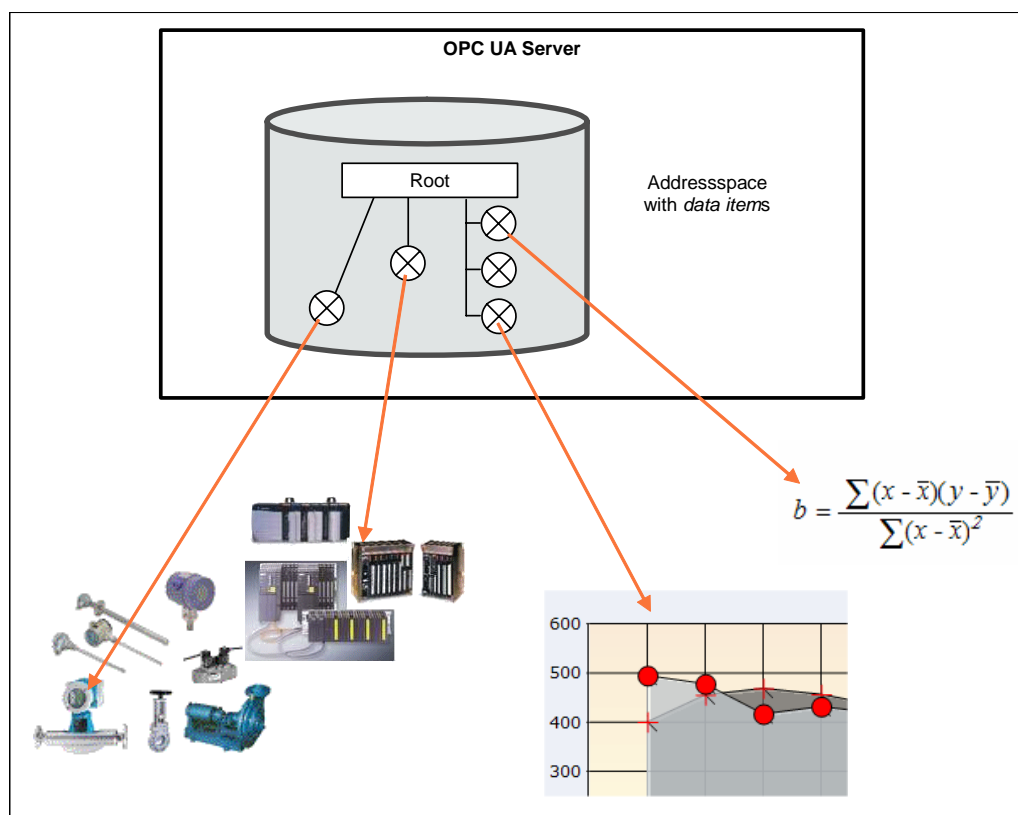


Figure 1 – OPC *DataItems* are linked to automation data

Clients may read or write *DataItems*, or monitor them for value changes. The services needed for these operations are specified in Part 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 *DataItem* type is the base type. *AnalogType* and *DiscreteType* (and its *TwoState* and *MultiState* subtypes) are specializations. Each of these *VariableTypes* can be further extended to form domain or server specific *DataItems*.

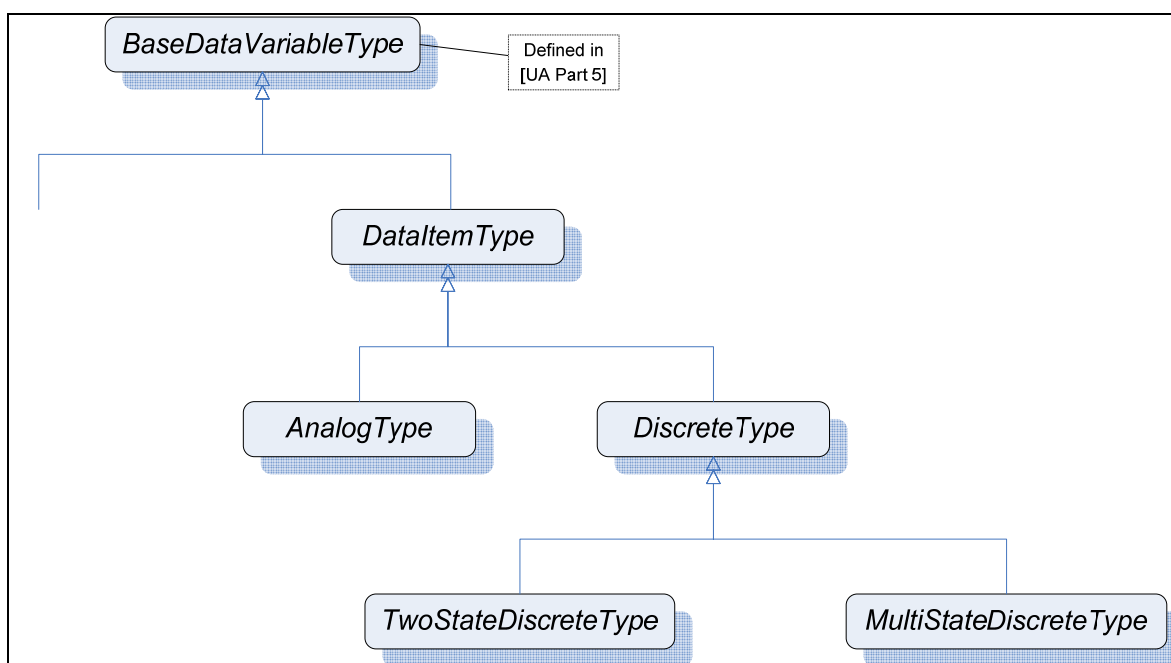


Figure 2 – DataItem VariableType Hierarchy

5.2 Variable Types

5.2.1 DataItem type

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

Table 1 – DataItem type Definition

Attribute	Value				
BrowseName	DataItem type				
IsAbstract	False				
ValueRank	-2 (-2 = 'Any')				
DataType	BaseDataType				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>BaseDataVariableType</i> defined in Part 5; i.e the <i>Properties</i> of that type are inherited.					
HasSubtype	VariableType	AnalogItem type	Defined in 5.2.2		
HasSubtype	VariableType	DiscreteItem type	Defined in 5.2.3		
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. E.g., a *ValuePrecision* of 20.000.000 defines a precision of 20 milliseconds.

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.

5.2.2 AnalogItem Type

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

Table 2 – AnalogItem Type Definition

Attribute	Value				
BrowseName	AnalogItem Type				
IsAbstract	False				
ValueRank	-2 (-2 = 'Any')				
DataType	Number				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>DataItem</i> Type defined in 5.2; i.e the <i>Properties</i> of that type are inherited.					
HasProperty	Variable	InstrumentRange	Range	PropertyType	Optional
HasProperty	Variable	EURange	Range	PropertyType	Mandatory
HasProperty	Variable	EngineeringUnits	EUInformation	PropertyType	Optional

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

Example: $InstrumentRange ::= \{-9999.9, 9999.9\}$

The *Range* type is specified in 5.5.2.

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 this range. Client software must be prepared to deal with this. Similarly a client may attempt to write a value that is outside 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: $EURange ::= \{-200.0, 1400.0\}$

See also 6.1 for a special monitoring filter (*PercentDeadband*) which is based on the engineering unit range.

EngineeringUnits specifies the units for the *DataItem*'s value (e.g., DEGC, hertz, seconds).

The *EUInformation* type is specified in 5.5.3.

If the item contains an array the *Properties* shall apply to all elements in the array.

5.2.3 DiscreteItemType

This *VariableType* is an abstract type. I.e., 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 3.

Table 3 – DiscreteItemType Definition

Attribute	Value				
BrowseName	DiscreteItemType				
IsAbstract	True				
ValueRank	-2 (-2 = 'Any')				
DataType	BaseDataType				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>DataItemType</i> defined in 5.2; i.e the <i>Properties</i> of that type are inherited.					
HasSubtype	VariableType	TwoStateDiscreteType	Defined in 5.2.3.1		
HasSubtype	VariableType	MultiStateDiscreteType	Defined in 5.2.3.2		

5.2.3.1 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 4.

Table 4 – TwoStateDiscreteType Definition

Attribute	Value				
BrowseName	TwoStateDiscreteType				
IsAbstract	False				
ValueRank	-2 (-2 = 'Any')				
DataType	Boolean				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>DiscreteItemType</i> defined in 5.2.3; i.e the <i>Properties</i> 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.

e.g. "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.

e.g. "STOP", "OPEN", "DISABLE", "UNSAFE", etc.

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

5.2.3.2 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 5.

Table 5 – MultiStateDiscreteType Definition

Attribute	Value				
BrowseName	MultiStateDiscreteType				
IsAbstract	False				
ValueRank	-2 (-2 = 'Any')				
DataType	UInteger				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>DiscreteItemType</i> defined in 5.2.3; i.e the <i>Properties</i> 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 the range of the list and robust servers should be prepared to handle writes of illegal values.

If the item contains an array 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 Part 3).

5.3 Address Space Model

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

Figure 3 illustrates the basic *AddressSpace* model of a *DataItem* – in this case an *AnalogItem*.

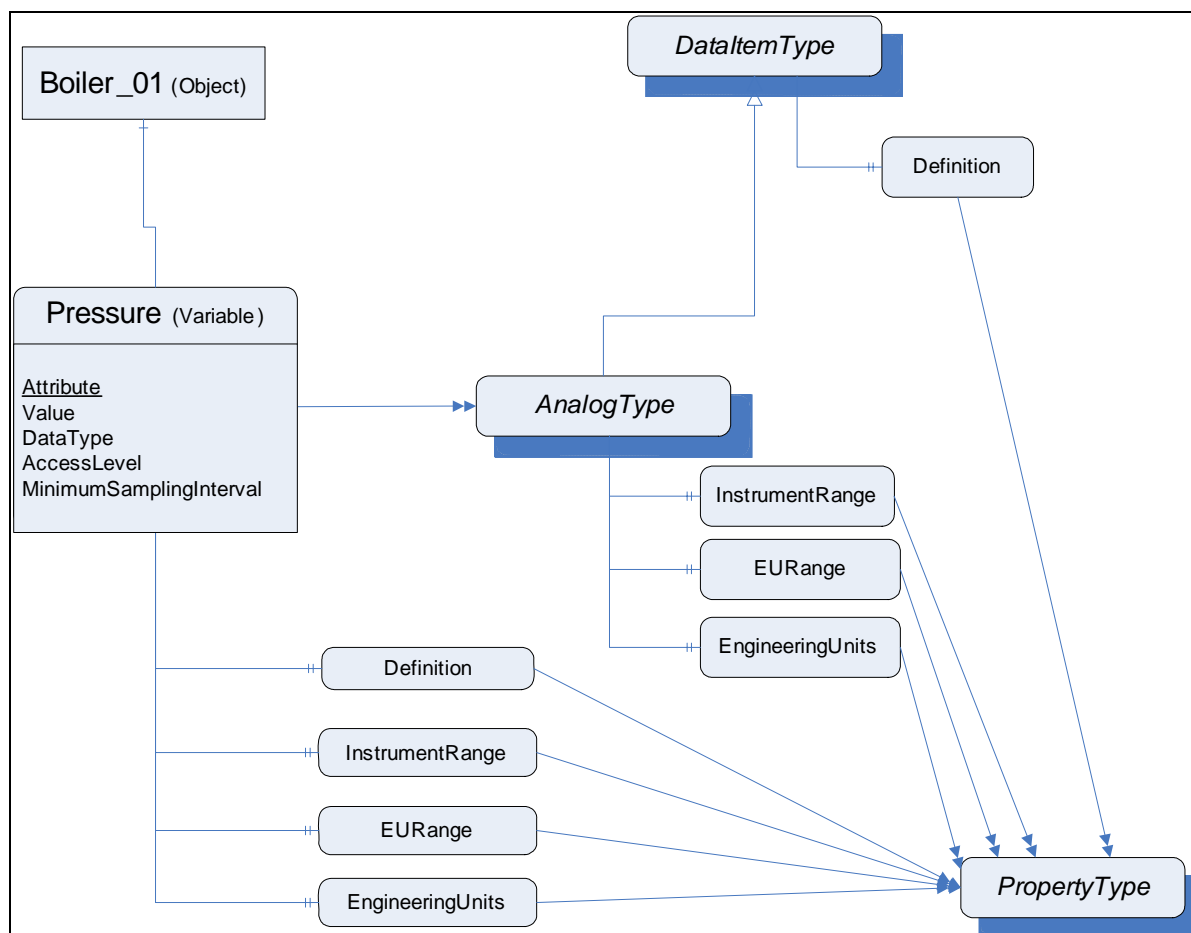


Figure 3 – Representation of *DataItems* in the *AddressSpace*

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 *AnalogType*). Additional characteristics of *DataItems* are defined using *Properties*. The *VariableTypes* in 5.2 specify which properties may exist. These *Properties* have been found useful for a wide range of Data Access clients. Servers that want to disclose similar information should use the OPC-defined *Property* rather than a vendor-specific one.

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

5.4 Attributes of DataItems

This section lists the *Attributes* of *Variables* that have particular importance for Data Access. They are specified in detail in Part 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 are.

When a client requests the *Value Attribute* for reading or monitoring, the server always returns a *StatusCode* (the quality and the server's ability to access/provide the value) and, optionally, a *ServerTimestamp* and a *SourceTimestamp*. See Part 4 for details on *StatusCode* and the meaning of the two timestamps. Specific status codes for Data Access are defined in 6.2.

5.5 Property DataTypes

5.5.1 Overview

Following is a description of the data types used for Data Access properties defined in this part.

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

5.5.2 Range

This structure defines the *Range* for a value. Its elements are defined in Table 6.

Table 6 – Range Data Type Structure

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

Its representation in the *AddressSpace* is defined in Table 7

Table 7 – Range Definition

Attributes	Value
BrowseName	Range

5.5.3 EUInformation

This structure contains information about the *EngineeringUnits*. Its elements are defined in Table 8.

The structure has been defined such that standards bodies can incorporate their engineering unit definitions into OPC UA. Servers shall use the *namespaceUri* in this structure to identify the proper organization.

Table 8 – EUInformation Data Type Structure

Name	Type	Description
EUInformation	structure	
namespaceUri	String	Identifies the organization (company, standards organization) that defines the <i>EUInformation</i> .
unitId	Int32	Identifier for programmatic evaluation. - 1 is used if a <i>unitId</i> is not available.
displayName	LocalizedText	The <i>displayName</i> of the engineering unit is typically the abbreviation of the engineering unit, e.g. "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 9

Table 9 – EUInformation Definition

Attributes	Value
BrowseName	EUInformation

6 Data Access specific usage of Services

Part 4 specifies the complete set of services. Those needed for the purpose of *DataAccess* are in particular:

- 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.

6.1 PercentDeadband

The *DataChangeFilter* in Part 4 defines the conditions under which a data change notification shall be reported. This filter contains a deadband which can be of type *AbsoluteDeadband* or *PercentDeadband*. Part 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* to generate an exception limit. An exception is determined as follows:

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

If the item is an array of values and any array element exceeds the *deadbandValue*, the entire monitored array is returned.

6.2 Data Access Status Codes

6.2.1 Overview

This section 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 Part 4. It includes a set of common operational result codes that also apply to Data Access.

6.2.2 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 slightly more generic than the description of data quality in the various fieldbus specifications.

In the following, Table 10 contains codes with BAD severity, indicating a failure;

Table 11 contains codes with UNCERTAIN severity, indicating that the value has been generated under sub-normal conditions.

Table 12 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 and are therefore defined in Part 4.

Table 10 – Bad operation level result codes

Symbolic Id	Description
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, 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	Communications to the data source is defined, but not established, and there is no last known value available. This status/substatus is used for cached values before the first value is received.
Bad_OutOfService	The source of the data is not operational.
Bad_DeadbandFilterInvalid	The specified <i>PercentDeadband</i> is not supported, since an <i>EURange</i> is not configured.

Table 11 – Uncertain operation level result codes

Symbolic Id	Description
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 <u>Good</u> sources.

Table 12 – Good operation level result codes

Symbolic Id	Description
Good_LocalOverride	The value has been Overridden. Typically this means the input has been disconnected and a manually-entered value has been "forced".

6.2.3 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 *DataItems* 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.

6.2.4 SemanticsChanged

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

OPC UA Servers that implement Data Access shall set this Bit in notifications if one or several of the following *Properties* changes:

- *EngineeringUnits* (could create problems if the client uses the value to perform calculations)
- *EURange* (could change the behaviour of a *Subscription* if a *PercentDeadband* filter is used)
- *FalseState*, *TrueState*, *EnumStrings* (changes can cause misinterpretation by users or (scripting) programs)

It should not be changed for any of the other Data Access *Properties*.

Clients should not process the data value until they re-read the mentioned *Properties* associated with the *Variable*.