

OPC Unified Architecture

Specification

Part 3: Address Space Model

AMENDMENT 1: Method Metadata

Release Candidate 1.04

April 4, 2018

|  |  |  |  |
| --- | --- | --- | --- |
| Specification Type: | Industry Standard Specification | Comments: | Report or view errata: http://www.opcfoundation.org/errata |
|  |  |  |  |
| Title: | OPC Unified Architecture  Part 3 :Address Space Model | Date: | April 4, 2018 |
|  |  |  |  |
| Version: | Release Candidate 1.04 | Software: | MS-Word |
|  |  | Source: | OPC UA Part 3 - Address Space Model Amendment 1 - Method Metadata RC 1.04.08.docx |
|  |  |  |  |
| Author: | OPC Foundation | Status: | Release Candidate |
|  |  |  |  |

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 © 2006-2018, 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  
H[TUhttp://www.opcfoundation.orgUT](http://www.opcfoundation.org)H.

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 FOUDATION 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 or 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: H[TUhttp://www.opcfoundation.org/errataUT](http://www.opcfoundation.org/errata)H

Revision 1.04 Amendment 1 Highlights

The following table includes the Mantis issues resolved by this Amendment.

|  |  |  |
| --- | --- | --- |
| **Mantis ID** | **Summary** | **Resolution** |
| [2765](https://www.opcfoundation.org/mantis/view.php?id=2765) | Support for optional method arguments or default parameters | Added DescribesArgument reference types to allow optional Metadata which can be used to identify optional arguments and default parameter values |
| [3792](https://www.opcfoundation.org/mantis/view.php?id=3792) | Support for additional information about method arguments | Added *DescribesArgument* reference types to allow optional additional *Metadata* to be included in the model |
| [4175](https://www.opcfoundation.org/mantis/view.php?id=4175) | ModelingRule for InputArguments, OutputArguments and Method Metadata | Added statements defining the use of Mandatory ModellingRule in Table 15 |

OPC Unified Architecture Specification

Part 3: Address Space Model

AMENDMENT 1: Method Metadata

Clause 5.7

Replace 5.7 with the following:

## Methods

### Method NodeClass

*Methods* define callable functions. *Methods* are invoked using the *Call* *Service* defined in [Part 4](http://www.opcfoundation.org/UA/Part4/). Method invocations are not represented in the *AddressSpace*. Method invocations always run to completion and always return responses when complete. *Methods* are defined using the *Method* *NodeClass*, specified in Table 15.

Table 15 – Method NodeClass

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Use | Data Type | Description |
| **Attributes** |  |  |  |
| Base NodeClass Attributes | M | -- | Inherited from the *Base NodeClass*. See [Part 3 Clause 5.2](http://www.opcfoundation.org/UA/Part3/). |
| Executable | M | Boolean | The *Executable* *Attribute* indicates if the *Method* is currently executable (“False” means not executable, “True” means executable).  The *Executable Attribute* does not take any user access rights into account, i.e. although the *Method* is executable this may be restricted to a certain user / user group. |
| UserExecutable | M | Boolean | The *UserExecutable* *Attribute* indicates if the *Method* is currently executable taking user access rights into account (“False” means not executable, “True” means executable). |
|  |  |  |  |
| **References** |  |  |  |
| HasProperty | 0..\* |  | *HasProperty* *References* identify the *Properties* for the *Method*. |
| HasModellingRule | 0..1 |  | *Methods* can point to at most one *ModellingRule* *Object* using a *HasModellingRule* *Reference* (see [Part 3 Clause 6.4.4](http://www.opcfoundation.org/UA/Part3/) for details on *ModellingRules*). |
| GeneratesEvent | 0..\* |  | *GeneratesEvent* *References* identify the type of *Events* that will be generated whenever the *Method* is called. |
| AlwaysGeneratesEvent | 0..\* |  | *AlwaysGeneratesEvent* *References* identify the type of *Events* that shall be generated whenever the *Method* is called. |
| DescribesArgument | 0..\* |  | *The DescribesArgument References are used to specify argument Metadata. If the SourceNode of this Reference is defined on an InstanceDeclaration the TargetNode shall use the Mandatory ModellingRule.* |
| <other References> | 0..\* |  | *Methods* may contain other *References*. |
|  |  |  |  |
| **Standard Properties** |  |  |  |
| NodeVersion | O | String | The *NodeVersion* *Property* is used to indicate the version of a *Node*.  The *NodeVersion* *Property* is updated each time a *Reference* is added or deleted to the *Node* the *Property* belongs to. *Attribute* value changes do not cause the *NodeVersion* to change. *Clients* may read the *NodeVersion Property* or subscribe to it to determine when the structure of a *Node* has changed. |
| InputArguments | O | Argument[] | The *InputArguments Property* is used to specify the arguments that shall be used by a client when calling the *Method. If this Property is defined on an InstanceDeclaration it shall use the Mandatory ModellingRule.* |
| OutputArguments | O | Argument[] | The *OutputArguments Property* specifies the result returned from the *Method* call. *If this Property is defined on an InstanceDeclaration it shall use the Mandatory ModellingRule.* |

The *Method NodeClass* inherits the base *Attributes* from the *Base NodeClass* defined in [Part 3 Clause 5.2](http://www.opcfoundation.org/UA/Part3/). The *Method NodeClass* defines no additional *Attributes*.

The *Executable* *Attribute* indicates whether the *Method* is executable, not taking user access rights into account. If the OPC UA *Server* cannot get the *Executable* information from the underlying system, it should state that it is executable. If a *Method* is called then the *Server* should transfer this request and return the corresponding *StatusCode* even if such a request is rejected. *StatusCodes* are defined in [Part 4](http://www.opcfoundation.org/UA/Part4/).

The *UserExecutable* *Attribute* indicates whether the *Method* is executable, taking user access rights into account. If the OPC UA *Server* cannot get any user rights related information from the underlying system, it should use the same value as used in the *Executable Attribute*. The *UserExecutable* *Attribute* can be set to “False”, even if the *Executable* *Attribute* is set to “True”, but it shall be set to “False” if the *Executable* *Attribute* is set to “False”. *Clients* cannot assume a *Method* can be executed based on the *UserExecutable* *Attribute*. It is possible that the *Server* may return an access denied error due to some *Server* specific change which was not reflected in the state of this *Attribute* at the time the *Client* accessed it.

*Properties* may be defined for *Methods* using *HasProperty* *References*. The *Properties* *InputArguments* and *OutputArguments* specify the input arguments and output arguments of the *Method*. Both contain an array of the *DataType* *Argument* as specified in 8.6. An empty array or a *Property* that is not provided indicates that there are no input arguments or output arguments for the *Method*.

The *Property* *NodeVersion* indicates the version of the *Method*. There are no additional *Properties* defined for *Methods* in this document. Additional parts of this series of standards may define additional *Properties* for *Methods*.

To specify its *ModellingRule*, a *Method* can use at most one *HasModellingRule Reference* pointing to a *ModellingRule* *Object*. *ModellingRules* are defined in [Part 3 Clause 6.4.4](http://www.opcfoundation.org/UA/Part3/).

*GeneratesEvent* *References* identify that *Methods* may generate an *Event* of the specified *EventType* or one of its subtypes for every call of the *Method*. A *Server* may generate one *Event* for each referenced *EventType* when a *Method* is successfully called.

*AlwaysGeneratesEvent* *References* identify that *Methods* will generate an *Event* of the specified *EventType* or one of its subtypes for every call of the *Method*. A *Server* shall always generate one *Event* for each referenced *EventType* when a *Method* is successfully called.

*Servers* should make *GeneratesEvent* *References* bidirectional *References*. However, it is allowed to be unidirectional when the *Server* is not able to expose the inverse direction pointing from the *EventType* to each *Method* generating the *EventType.*

*GeneratesEvent* *References* are optional, i.e. the call of a *Method* may produce *Events* of an *EventType* that is not referenced with a *GeneratesEvent* *Reference* by the *Method*.

*Methods* may use additional *References* to define relationships to other *Nodes*. No restrictions are placed on the types of *References* used or on the *NodeClasses* of the *Nodes* that may be referenced. However, restrictions may be defined by the *ReferenceType* excluding its use for *Methods*. Standard *ReferenceTypes* are described in [Part 3 Clause 7](http://www.opcfoundation.org/UA/Part3/).

A *Method* shall always be the *TargetNode* of at least one *HasComponent* *Reference*. The *SourceNode* of these *HasComponent* *References* shall be an *Object* or an *ObjectType*. If a *Method* is called then the *NodeId* of one of those *Nodes* shall be put into the Call *Service* defined in [Part 4](http://www.opcfoundation.org/UA/Part4/) as parameter to detect the context of the *Method* operation.

If the *Method* is used as *InstanceDeclaration* (see [Part 3 Clause 4.5](http://www.opcfoundation.org/UA/Part3/)) all *Nodes* referenced with forward *hierarchical References* shall have unique *BrowseNames* in the context of this *Method*.

The *Variable* referenced by a *DescribesArgument* *ReferenceType* shall use a *BrowseName* equal to the name of the *Argument* it describes. The *NameSpace* of the *BrowseName* shall be ignored by a Client when performing an equality check with an Argument name. The *Variable* referenced by the *DescribesArgument* shall have the same *DataType* as the *Argument’s* *DataType*.

*Argument* names shall be unique within the scope of the *Method* when optional *DescribesArgument* *References* are used.

An example use of the *DescribesArgument* *Reference* is illustrated in Figure 8A. In this example an *ObjectType* defines a *Method* which illustrates the following:

* Output1 as a discrete output argument with a *DescribesArgument* *Reference* to a *TwoStateDescreteType* *Variable* which provides descriptions of the states of the output argument.
* Input1 as a numeric input argument with a *DescribesArgument* *Reference* to a *DataVariable* *Variable* which provides the default value 42.
* Input2 as a numeric input argument with a *DescribesArgument* *Reference* to an *AnalogItemType* *Variable* which provides an engineering units range of 0 kPa to 100 kPa.
* Input3 as an optional discrete input argument with a *DescribesArgument* *Reference* to a *TwoStateDescreteType* *Variable* which provides descriptions of the states of the input argument and a default value of true.
* Object1 as an instance of MyObjectType with an instance specific *DescribesArgument* *Reference* to an instance specific *AnalogItemType* *Variable* which provides an instance specific engineering range of 0 kPa to 200 kPa and a default value of 75 kPa for the Input2 argument.



Figure 8A – Method Metadata Example

### DescribesArgument ReferenceType

The *DescribesArgument ReferenceType* is an abstract *ReferenceType* that is a subtype of the *HasComponent ReferenceType*.

The semantic of the *DescribesArgument ReferenceType* – extends the semantic of the *HasComponent ReferenceType* to reference argument Meta data of a Method NodeClass.

The TargetNode of this ReferenceType shall be a Variable.

### DescribesOutputArgument ReferenceType

The Describes*OutputArgument ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *DescribesArgument ReferenceType*.

The semantic of the Describes*OutputArgument ReferenceType* – extends the semantic of the *DescribesArgument ReferenceType* to reference output arguments of a Method NodeClass.

There are no additional constraints defined for this *ReferenceType*.

### DescribesInputArgument ReferenceType

The Describes*InputArgument ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *DescribesArgument ReferenceType*.

The semantic of the Describes*InputArgument ReferenceType* – extends the semantic of the *DescribesArgument ReferenceType* to reference input arguments of a Method NodeClass.

There are no additional constraints defined for this *ReferenceType*.

### DescribesOptionalInputArgument ReferenceType

The Describes*OptionalInputArgument ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the Describes*InputArgument ReferenceType*.

The semantic of the Describes*OptionalInputArgument ReferenceType* – extends the semantic of the Describes*InputArgument ReferenceType* to reference optional input arguments of a Method NodeClass. Optional input arguments shall always follow any non-optional input arguments in the *InputArguments* array. For example if a method has 3 arguments with 1 being optional then the 3rd argument shall be the optional one.

There are no additional constraints defined for this *ReferenceType*.

\_\_\_\_\_\_\_\_\_\_\_\_