

DIN EN IEC 62541-3

ICS 25.040.40; 35.200; 35.240.50

Ersatz für
DIN EN 62541-3:2015-11
Siehe Anwendungsbeginn

**OPC Unified Architecture –
Teil 3: Adressraummodell
(IEC 62541-3:2020);
Englische Fassung EN IEC 62541-3:2020**

OPC Unified Architecture –
Part 3: Address Space Model
(IEC 62541-3:2020);
English version EN IEC 62541-3:2020

Architecture unifiée OPC –
Partie 3: Modèle d'espace d'adressage
(IEC 62541-3:2020);
Version anglaise EN IEC 62541-3:2020

Gesamtumfang 126 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2020-08-12 angenommene Europäische Norm als DIN-Norm ist 2021-08-01.

Für DIN EN 62541-3:2015-11 besteht eine Übergangsfrist bis 2023-08-12.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN EN IEC 62541-3:2019-09.

Für dieses Dokument ist das nationale Arbeitsgremium DKE/K 931 „Systemaspekte der Automatisierung“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE (www.dke.de) zuständig.

Aktuelle Informationen zu diesem Dokument können über die Internetseiten von DKE (www.dke.de) und DIN (www.din.de) durch eine Suche nach der Dokumentennummer aufgerufen werden.

Die enthaltene IEC-Publikation wurde vom IEC/SC 65E „Devices and integration in enterprise systems“ erarbeitet und unterliegt dem Copyright der IEC .

Das IEC-Komitee hat entschieden, dass der Inhalt dieses Dokuments bis zu dem Datum (stability date) unverändert bleiben soll, das auf der IEC-Website unter „<http://webstore.iec.ch>“ zu diesem Dokument angegeben ist. Zu diesem Zeitpunkt wird entsprechend der Entscheidung des Komitees das Dokument

- bestätigt,
- zurückgezogen,
- durch eine Folgeausgabe ersetzt oder
- geändert.

In diesem Dokument und in den anderen Teilen der IEC 62541-Reihe werden bestimmte Dokumentenkonventionen verwendet:

Kursivschrift wird verwendet, um einen definierten Begriff oder eine Definition zu kennzeichnen, der/die in dem Abschnitt „Begriffe und Abkürzungen“ in einem der Teile der IEC 62541-Reihe erscheint.

Kursivschrift wird auch verwendet, um den Namen eines Service-Eingangs- oder -Ausgangsparameters oder den Namen einer Struktur oder eines Elements einer Struktur zu kennzeichnen, die normalerweise in Tabellen definiert sind.

Die *kursiv* gedruckten Begriffe und Namen werden mit wenigen Ausnahmen in Camel-Case-Schreibweise geschrieben (die Schreibweise von zusammengesetzten Wörtern oder Phrasen, in denen die Elemente ohne Leerzeichen verbunden sind, wobei der Anfangsbuchstabe jedes Elements innerhalb der Verbindung groß geschrieben wird). Zum Beispiel lautet der definierte Begriff „AddressSpace“ statt „Address Space“. Dies macht es einfacher zu verstehen, dass es eine einzige Definition für AddressSpace gibt und nicht separate Definitionen für Address und Space.

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf ein Dokument ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils aktuellste Ausgabe des in Bezug genommenen Dokuments.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe des Dokuments.

Der Zusammenhang der zitierten Dokumente mit den entsprechenden deutschen Dokumenten ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation.

Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

Das Präsidium des DIN hat mit Präsidialbeschluss 1/2004 festgelegt, dass DIN-Normen, deren Inhalt sich auf internationale Arbeitsergebnisse der Informationsverarbeitung gründet, unter bestimmten Bedingungen allein in englischer Sprache veröffentlicht werden dürfen. Diese Bedingungen sind für die vorliegende Norm erfüllt.

Da sich die Benutzer der vorliegenden Norm der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN IEC 62541-3 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt und Abschnitt 3 (außer Unterabschnitte 3.2 und 3.3), der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA der vorliegenden Norm hinzugefügt. Für die meisten der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Diese Norm steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

Das Original-Dokument enthält Bilder in Farbe, die in der Papierversion in einer Graustufen-Darstellung wiedergegeben werden. Elektronische Versionen dieses Dokuments enthalten die Bilder in der originalen Farbdarstellung.

Änderungen

Gegenüber DIN EN 62541-3:2015-11 wurden folgende Änderungen vorgenommen:

- a) neuen verbesserten Ansatz zur Anzeige von Strukturdefinitionen hinzugefügt. Ein Attribut eines DataType Knoten enthält nun einfach eine binäre Beschreibung;
- b) neue Flags für Variablen zur Angabe der Atomarität beim Lesen oder Schreiben hinzugefügt;
- c) Rollen und Zulassungen zur Ermöglichung der Konfiguration einer rollenbasierten Autorisierung hinzugefügt;
- d) neue Datentypen hinzugefügt: „Union“, „Decimal“, „OptionSet“, „DateString“, „TimeString“, „DurationString“, „NormalizedString“, „DecimalString“ und „AudioDataType“;
- e) Definition zur Anwendung der ModellingRules OptionalPlaceHolder und MandatoryPlaceHolder für Verfahren hinzugefügt;
- f) optionale Eigenschaften „MaxCharacters“ und „MaxByteStringLength“ zu variablen Knoten hinzugefügt.

Frühere Ausgaben

DIN EN 62541-3: 2011-09, 2015-11

Nationaler Anhang NA (informativ)

Auszugsweise Übersetzung

Reihenfolge und Inhalt der folgenden Abschnitte sind identisch mit denen in den jeweiligen Abschnitten der Englischen Fassung.

1 Anwendungsbereich

Dieser Teil der Normenreihe IEC 62541 legt den *AdressRaum* (en: *AddressSpace*) und seine *Objekte* (en: *Objects*) der OPC Unified Architecture (OPC UA) (vereinheitlichte Architektur der plattformunabhängigen Kommunikation) fest. Dieses Dokument stellt das OPC-UA-Metamodell dar, auf welchem OPC-UA-Informationsmodelle beruhen.

3 Begriffe, Abkürzungen und Konventionen

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach *IEC TR 62541-1* und die folgenden Begriffe.

ISO und IEC stellen terminologische Datenbanken für die Anwendung in der Normung unter den folgenden Adressen bereit:

- IEC Electropedia: verfügbar unter <http://www.electropedia.org/>
- ISO Online Browsing Platform: verfügbar unter <http://www.iso.org/obp>

3.1.1

DatenTyp

(en: *DataType*)

Instanz eines *DatenTyp Knoten* (en: *DataType Node*), die gemeinsam mit dem *Attribut* (en: *Attribute*) *WertRang* (en: *ValueRank*) zur Festlegung des Datentyps einer *Variablen* verwendet wird

3.1.2

DatenTypeID

(en: *DataTypeID*)

KnotenID (Identität eines Knoten) (en: *NodeID*) eines *DatenTyp Knotens*

3.1.3

DatenVariable

(en: *DataVariable*)

Variable (en: *Variable*), die den Wert eines *Objekt* (en: *Object*), entweder direkt oder indirekt für komplexe *Variablen* darstellt; dabei sind *Variablen* stets der *ZielKnoten* (en: *TargetNode*) einer *BesitztKomponente Referenz* (en: *HasComponent Reference*)

3.1.4

EreignisTyp

(en: *EventType*)

ObjektTyp (en: *ObjectType*) *Knoten*, der die Typdefinition eines *Ereignisses* (en: *Event*) darstellt

3.1.5

hierarchische Referenz

(en: *Hierarchical Reference*)

Referenz, welche zum Aufbau von Hierarchien im *AdressRaum* (en: *AddressSpace*) verwendet wird

Anmerkung 1 zum Begriff: Alle hierarchischen ReferenzTypen (en: ReferenceTypes) werden von HierarchischenReferenzen (en: HierarchicalReferences) abgeleitet.

3.1.6

InstanzDeklaration

(en: InstanceDeclaration)

Knoten, der von einem komplexen *TypDefinitionsKnoten* (en: TypeDefinitionNode) zur Deklaration seiner komplexen Struktur verwendet wird

Anmerkung 1 zum Begriff: Es ist eine Instanz, die von einer Typdefinition verwendet wird.

3.1.7

ModellierungsRegel

(en: ModellingRule)

Metadaten einer *InstanzDeklaration*, die festlegt, wie die *InstanzDeklaration* für die Instantiierung zu verwenden ist; sie legt auch Untertypenregeln für eine *InstanzDeklaration* fest

3.1.8

Eigenschaft

(en: Property)

Variable, die der *ZielKnoten* (en: TargetNode) für eine *Referenz BesitztEigenschaft* (en: HasProperty) ist

Anmerkung 1 zum Begriff: Eigenschaften beschreiben die Merkmale eines Knoten.

3.1.9

QuellKnoten

(en: SourceNode)

Knoten, der eine *Referenz* zu einem weiteren *Knoten* hat

BEISPIEL: In der *Referenz* „A enthält B“ ist „A“ der *QuellKnoten*.

3.1.10

ZielKnoten

(en: TargetNode)

Knoten, auf den von einem weiteren *Knoten* Bezug genommen wird

BEISPIEL: In der *Referenz* „A enthält B“ ist „B“ der *ZielKnoten*.

3.1.11

TypDefinitionsKnoten

(en: TypeDefinitionNode)

Knoten, der zur Festlegung des Typs eines weiteren *Knoten* verwendet wird

Anmerkung 1 zum Begriff: Die Knoten *ObjektTyp* (en: ObjectType) und *VariablenTyp* sind *TypDefinitionsKnoten*.

3.1.12

VariablenTyp

(en: VariableType)

Knoten, der die Typdefinition für eine *Variable* darstellt

– Leerseite –

English Version

OPC Unified Architecture - Part 3: Address Space Model
(IEC 62541-3:2020)

Architecture unifiée OPC - Partie 3: Modèle d'espace
d'adressage
(IEC 62541-3:2020)

OPC Unified Architecture - Teil 3: Adressraummodell
(IEC 62541-3:2020)

This European Standard was approved by CENELEC on 2020-08-12. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.



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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

European foreword

The text of document 65E/715/FDIS, future edition 3 of IEC 62541-3, prepared by SC 65E "Devices and integration in enterprise systems" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN IEC 62541-3:2020.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by (dop) 2021-05-12
publication of an identical national standard or by endorsement
- latest date by which the national standards conflicting with the (dow) 2023-08-12
document have to be withdrawn

This document supersedes EN 62541-3:2015 and all of its amendments and corrigenda (if any).

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

This document has been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association.

Endorsement notice

The text of the International Standard IEC 62541-3:2020 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following note has to be added for the standard indicated:

IEC 62541-11 NOTE Harmonized as EN 62541-11.

Annex ZA (normative)

Normative references to international publications with their corresponding European publications

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.

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

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

<u>Publication</u>	<u>Year</u>	<u>Title</u>	<u>EN/HD</u>	<u>Year</u>
IEC/TR 62541-1	–	OPC unified architecture – Part 1: Overview and concepts	CLC/TR 62541-1	–
IEC 62541-4	–	OPC Unified Architecture – Part 4: Services	–	–
IEC 62541-5	–	OPC Unified Architecture – Part 5: Information Model	–	–
IEC 62541-6	–	OPC Unified Architecture – Part 6: Mappings	–	–
IEC 62541-8	–	OPC Unified Architecture – Part 8: Data Access	EN IEC 62541-8	–
ISO/IEC/IEEE 60559	2011	Information technology – Microprocessor Systems – Floating-Point arithmetic	–	–
ISO 639	series	Code for the representation of names of languages	–	–
ISO 3166	series	Codes for the representation of names of countries	–	–
ISO 8601	series		–	–
IETF RFC 5646	–	Tags for Identifying Languages	–	–
Unicode Standard Annex #15	–	Unicode Normalization Forms	–	–
W3C XML Schema Definition Language (XSD) Part 2	–	Data Types	–	–
TAI	–	International Atomic Time	–	–

CONTENTS

European foreword	8
Annex ZA (normative) Normative references to international publications with their corresponding European publications	9
1 Scope.....	20
2 Normative references	20
3 Terms, definitions, abbreviated terms and conventions.....	20
3.1 Terms and definitions	20
3.2 Abbreviated terms	22
3.3 Conventions.....	22
3.3.1 Conventions for AddressSpace figures	22
3.3.2 Conventions for defining NodeClasses	22
4 AddressSpace concepts.....	23
4.1 Overview.....	23
4.2 Object Model	24
4.3 Node Model	24
4.3.1 General.....	24
4.3.2 NodeClasses	25
4.3.3 Attributes	25
4.3.4 References	25
4.4 Variables.....	26
4.4.1 General.....	26
4.4.2 Properties	26
4.4.3 DataVariables	26
4.5 TypeDefinitionNodes	27
4.5.1 General.....	27
4.5.2 Complex TypeDefinitionNodes and their InstanceDeclarations	27
4.5.3 Subtyping.....	28
4.5.4 Instantiation of complex TypeDefinitionNodes	28
4.6 Event Model.....	29
4.6.1 General.....	29
4.6.2 EventTypes.....	30
4.6.3 Event Categorization	30
4.7 Methods.....	31
4.8 Roles.....	31
4.8.1 Overview.....	31
4.8.2 Well-known Roles	32
4.8.3 Evaluating Permissions with Roles	32
5 Standard NodeClasses.....	35

5.1	Overview	35
5.2	Base NodeClass.....	35
5.2.1	General.....	35
5.2.2	NodeId.....	35
5.2.3	NodeClass.....	35
5.2.4	BrowseName.....	36
5.2.5	DisplayName.....	36
5.2.6	Description	36
5.2.7	WriteMask	36
5.2.8	UserWriteMask.....	37
5.2.9	RolePermissions	37
5.2.10	UserRolePermissions.....	38
5.2.11	AccessRestrictions	38
5.3	ReferenceType NodeClass	38
5.3.1	General.....	38
5.3.2	Attributes	39
5.3.3	References	41
5.4	View NodeClass.....	41
5.5	Objects	43
5.5.1	Object NodeClass	43
5.5.2	ObjectType NodeClass	45
5.5.3	Standard ObjectType FolderType.....	46
5.5.4	Client-side creation of Objects of an ObjectType.....	47
5.6	Variables	47
5.6.1	General.....	47
5.6.2	Variable NodeClass	47
5.6.3	Properties	52
5.6.4	DataVariable.....	52
5.6.5	VariableType NodeClass	53
5.6.6	Client-side creation of Variables of an VariableType.....	55
5.7	Method NodeClass.....	55
5.8	DataTypes	57
5.8.1	DataType Model.....	57
5.8.2	Encoding rules for different kinds of DataTypes	58
5.8.3	DataType NodeClass	59
5.8.4	DataTypeEncoding and encoding information	61
5.9	Summary of Attributes of the NodeClasses	62
6	Type Model for ObjectTypes and VariableTypes.....	63
6.1	Overview	63

6.2	Definitions	63
6.2.1	InstanceDeclaration	63
6.2.2	Instances without ModellingRules	63
6.2.3	InstanceDeclarationHierarchy	63
6.2.4	Similar Node of InstanceDeclaration	63
6.2.5	BrowsePath	64
6.2.6	Attribute Handling of InstanceDeclarations	64
6.2.7	Attribute Handling of Variable and VariableTypes	64
6.2.8	NodeIds of InstanceDeclarations	64
6.3	Subtyping of ObjectTypes and VariableTypes	64
6.3.1	Overview	64
6.3.2	Attributes	64
6.3.3	InstanceDeclarations	65
6.4	Instances of ObjectTypes and VariableTypes	69
6.4.1	Overview	69
6.4.2	Creating an Instance	69
6.4.3	Constraints on an Instance	70
6.4.4	ModellingRules	70
6.5	Changing type definitions that are already used	78
7	Standard ReferenceTypes	78
7.1	General	78
7.2	References ReferenceType	79
7.3	HierarchicalReferences ReferenceType	79
7.4	NonHierarchicalReferences ReferenceType	80
7.5	HasChild ReferenceType	80
7.6	Aggregates ReferenceType	80
7.7	HasComponent ReferenceType	80
7.8	HasProperty ReferenceType	80
7.9	HasOrderedComponent ReferenceType	81
7.10	HasSubtype ReferenceType	81
7.11	Organizes ReferenceType	81
7.12	HasModellingRule ReferenceType	81
7.13	HasTypeDefinition ReferenceType	82
7.14	HasEncoding ReferenceType	82
7.15	GeneratesEvent	82
7.16	AlwaysGeneratesEvent	82
7.17	HasEventSource	82
7.18	HasNotifier	83
8	Standard DataTypes	84

8.1	General.....	84
8.2	NodeId	84
8.2.1	General.....	84
8.2.2	NamespaceIndex	85
8.2.3	IdentifierType.....	85
8.2.4	Identifier value	85
8.3	QualifiedName.....	86
8.4	LocaleId	86
8.5	LocalizedText	87
8.6	Argument.....	87
8.7	BaseDataType.....	88
8.8	Boolean	88
8.9	Byte	88
8.10	ByteString.....	88
8.11	DateTime.....	88
8.12	Double	89
8.13	Duration	89
8.14	Enumeration	89
8.15	Float	89
8.16	Guid	89
8.17	SByte.....	89
8.18	IdType	89
8.19	Image	89
8.20	ImageBMP.....	89
8.21	ImageGIF	89
8.22	ImageJPG	90
8.23	ImagePNG.....	90
8.24	Integer	90
8.25	Int16	90
8.26	Int32	90
8.27	Int64	90
8.28	TimeZoneDataType	90
8.29	NamingRuleType.....	90
8.30	NodeClass.....	91
8.31	Number.....	91
8.32	String	91
8.33	Structure.....	91
8.34	UInteger.....	91
8.35	UInt16.....	91

8.36	UInt32	91
8.37	UInt64	91
8.38	UtcTime	92
8.39	XmlElement	92
8.40	EnumValueType	92
8.41	OptionSet	92
8.42	Union	93
8.43	DateString	93
8.44	DecimalString	93
8.45	DurationString	93
8.46	NormalizedString	94
8.47	TimeString	94
8.48	DataTypeDefinition	95
8.49	StructureDefinition	95
8.50	EnumDefinition	95
8.51	StructureField	96
8.52	EnumField	96
8.53	AudioDataType	97
8.54	Decimal	97
8.55	PermissionType	97
8.56	AccessRestrictionsType	98
8.57	AccessLevelType	99
8.58	AccessLevelExType	99
8.59	EventNotifierType	100
8.60	AttributeWriteMask	100
9	Standard EventTypes	101
9.1	General	101
9.2	BaseEventType	102
9.3	SystemEventType	102
9.4	ProgressEventType	102
9.5	AuditEventType	103
9.6	AuditSecurityEventType	104
9.7	AuditChannelEventType	104
9.8	AuditOpenSecureChannelEventType	104
9.9	AuditSessionEventType	104
9.10	AuditCreateSessionEventType	104
9.11	AuditUrlMismatchEventType	105
9.12	AuditActivateSessionEventType	105
9.13	AuditCancelEventType	105

9.14	AuditCertificateEventType	105
9.15	AuditCertificateDataMismatchEventType	105
9.16	AuditCertificateExpiredEventType	105
9.17	AuditCertificateInvalidEventType	105
9.18	AuditCertificateUntrustedEventType	105
9.19	AuditCertificateRevokedEventType	105
9.20	AuditCertificateMismatchEventType	106
9.21	AuditNodeManagementEventType	106
9.22	AuditAddNodesEventType	106
9.23	AuditDeleteNodesEventType	106
9.24	AuditAddReferencesEventType	106
9.25	AuditDeleteReferencesEventType	106
9.26	AuditUpdateEventType	106
9.27	AuditWriteUpdateEventType	106
9.28	AuditHistoryUpdateEventType	106
9.29	AuditUpdateMethodEventType	106
9.30	DeviceFailureEventType	106
9.31	SystemStatusChangeEventType	107
9.32	ModelChangeEvents	107
9.32.1	General	107
9.32.2	NodeVersion Property	107
9.32.3	Views	107
9.32.4	Event compression	107
9.32.5	BaseModelChangeEventType	107
9.32.6	GeneralModelChangeEventType	108
9.32.7	Guidelines for ModelChangeEvents	108
9.33	SemanticChangeEventType	108
9.33.1	General	108
9.33.2	ViewVersion and NodeVersion Properties	108
9.33.3	Views	108
9.33.4	Event compression	108
Annex A (informative)	How to use the Address Space Model	109
A.1	Overview	109
A.2	Type definitions	109
A.3	ObjectTypes	109
A.4	VariableTypes	109
A.4.1	General	109
A.4.2	Properties or DataVariables	110
A.4.3	Many Variables and/or structured DataTypes	110

A.5	Views	110
A.6	Methods	111
A.7	Defining ReferenceTypes	111
A.8	Defining ModellingRules	111
Annex B (informative) OPC UA Meta Model in UML		112
B.1	Background	112
B.2	Notation	112
B.3	Meta Model	114
B.3.1	Base	114
B.3.2	ReferenceType	114
B.3.3	Predefined ReferenceTypes	116
B.3.4	Attributes	116
B.3.5	Object and ObjectType	117
B.3.6	EventNotifier	118
B.3.7	Variable and VariableType	118
B.3.8	Method	119
B.3.9	DataType	120
B.3.10	View	121
Annex C (normative) Graphical notation		122
C.1	General	122
C.2	Notation	122
C.2.1	Overview	122
C.2.2	Simple notation	122
C.2.3	Extended notation	124
Bibliography		126
Figure 1 – AddressSpace Node diagrams		22
Figure 2 – OPC UA Object Model		24
Figure 3 – AddressSpace Node Model		25
Figure 4 – Reference Model		26
Figure 5 – Example of a Variable defined by a VariableType		27
Figure 6 – Example of a Complex TypeDefinition		28
Figure 7 – Object and its Components defined by an ObjectType		29
Figure 8 – Permissions in the Address Space		38
Figure 9 – Symmetric and Non-Symmetric References		40
Figure 10 – Variables, VariableTypes and their DataTypes		58
Figure 11 – DataType Model		58
Figure 12 – Example of DataType Modelling		62
Figure 13 – Subtyping TypeDefinitionNodes		65

Figure 14 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType	68
Figure 15 – An Instance and its TypeDefinitionNode	69
Figure 16 – Example of several References between InstanceDeclarations.....	70
Figure 17 – Example of changing instances based on InstanceDeclarations.....	72
Figure 18 – Example of changing InstanceDeclarations based on an InstanceDeclaration	73
Figure 19 – Use of the Standard ModellingRule Mandatory	74
Figure 20 – Example using the Standard ModellingRules Optional and Mandatory.....	75
Figure 21 – Example of using ExposesItsArray	76
Figure 22 – Complex example of using ExposesItsArray	76
Figure 23 – Example using OptionalPlaceholder with an Object and Variable.....	76
Figure 24 – Example using OptionalPlaceholder with a Method.....	77
Figure 25 – Example of using MandatoryPlaceholder for Object and Variable	78
Figure 26 – Standard ReferenceType Hierarchy	79
Figure 27 – Event Reference Example	83
Figure 28 – Complex Event Reference Example	84
Figure 29 – Standard EventType Hierarchy	102
Figure 30 – Audit Behaviour of a Server	103
Figure 31 – Audit Behaviour of an Aggregating Server.....	104
Figure B.1 – Background of OPC UA Meta Model.....	112
Figure B.2 – Notation (I)	113
Figure B.3 – Notation (II)	113
Figure B.4 – Base	114
Figure B.5 – Reference and ReferenceType.....	115
Figure B.6 – Predefined ReferenceTypes	116
Figure B.7 – Attributes.....	117
Figure B.8 – Object and ObjectType	118
Figure B.9 – EventNotifier	118
Figure B.10 – Variable and VariableType	119
Figure B.11 – Method	120
Figure B.12 – DataType	120
Figure B.13 – View	121
Figure C.1 – Example of a Reference connecting two Nodes	123
Figure C.2 – Example of using a TypeDefinition inside a Node.....	124
Figure C.3 – Example of exposing Attributes	125
Figure C.4 – Example of exposing Properties inline	125
Table 1 – NodeClass Table Conventions.....	22
Table 2 – Well-known Roles.....	32
Table 3 – Example Roles	33

Table 4 – Example Nodes	33
Table 5 – Example Role assignment	34
Table 6 – Examples of evaluating access	34
Table 7 – Base NodeClass	35
Table 8 – RolePermissionType.....	37
Table 9 – ReferenceType NodeClass.....	39
Table 10 – View NodeClass	42
Table 11 – Object NodeClass.....	43
Table 12 – ObjectType NodeClass.....	45
Table 13 – Variable NodeClass	47
Table 14 – VariableType NodeClass.....	53
Table 15 – Method NodeClass	56
Table 16 – DataType NodeClass.....	59
Table 17 – Overview of Attributes.....	62
Table 18 – The InstanceDeclarationHierarchy for BetaType	66
Table 19 – The Fully-Inherited InstanceDeclarationHierarchy for BetaType.....	67
Table 20 – Rule for ModellingRules Properties when Subtyping	71
Table 21 – Properties of ModellingRules.....	73
Table 22 – NodeId Definition	84
Table 23 – IdentifierType Values	85
Table 24 – NodeId Null Values	86
Table 25 – QualifiedName Definition	86
Table 26 – LocaleId Examples	87
Table 27 – LocalizedText Definition.....	87
Table 28 – Argument Definition	88
Table 29 – TimeZoneDataType Definition	90
Table 30 – NamingRuleType Values	90
Table 31 – NodeClass Values	91
Table 32 – EnumValueType Definition	92
Table 33 – OptionSet Definition.....	93
Table 34 – StructureDefinition Structure	95
Table 35 – EnumDefinition Structure.....	95
Table 36 – StructureField Structure.....	96
Table 37 – EnumField Structure	96
Table 38 – PermissionType Definition.....	97
Table 39 – AccessRestrictionsType Definition	99
Table 40 – AccessLevelType Definition	99
Table 41 – AccessLevelExType Definition	100
Table 42 – EventNotifierType Definition	100

Table 43 – Bit mask for WriteMask and UserWriteMask.....	101
Table C.1 – Notation of Nodes depending on the NodeClass	123
Table C.2 – Simple Notation of Nodes depending on the NodeClass	123

OPC UNIFIED ARCHITECTURE –

Part 3: Address Space Model

1 Scope

This part of IEC 62541 defines the OPC Unified Architecture (OPC UA) *AddressSpace* and its *Objects*. This document is the OPC UA meta model on which OPC UA information models are based.

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 62541-4, *OPC Unified Architecture – Part 4: Services*

IEC 62541-5:–, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

IEC 62541-8, *OPC Unified Architecture – Part 8: Data Access*

ISO/IEC/IEEE 60559:2011, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

ISO 639 (all parts), *Codes for the representation of names of languages*

ISO 3166 (all parts), *Codes for the representation of names of countries and their subdivisions*

ISO 8601 (all parts), *Date and time – Representations for information interchange*

IETF RFC 5646, *Tags for Identifying Languages*
<http://tools.ietf.org/html/rfc5646>

Unicode Standard Annex #15: *Unicode Normalization Forms*,
<http://www.unicode.org/reports/tr15/>

W3C XML Schema Definition Language (XSD) Part 2: *DataTypes*
<http://www.w3.org/TR/xmlschema-2/>

TAI: *International Atomic Time*
<http://www.bipm.org/en/bipm-services/timescales/tai.html>

3 Terms, definitions, abbreviated terms and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1 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

DataType

instance of a *DataType Node* that is used together with the *ValueRank Attribute* to define the data type of a *Variable*

3.1.2

DataTypeId

NodeId of a *DataType Node*

3.1.3

DataVariable

Variable that represents the *value* of an *Object*, either directly or indirectly for complex *Variables*, where the *Variables* are always the *TargetNode* of a *HasComponent Reference*

3.1.4

EventType

ObjectType Node that represents the type definition of an *Event*

3.1.5

Hierarchical Reference

Reference that is used to construct hierarchies in the *AddressSpace*

Note 1 to entry: All hierarchical ReferenceTypes are derived from HierarchicalReferences.

3.1.6

InstanceDeclaration

Node that is used by a complex *TypeDefinitionNode* to expose its complex structure

Note 1 to entry: This is an instance used by a type definition.

3.1.7

ModellingRule

metadata of an *InstanceDeclaration* that defines how the *InstanceDeclaration* will be used for instantiation and also defines subtyping rules for an *InstanceDeclaration*

3.1.8

Property

Variable that is the *TargetNode* for a *HasProperty Reference*

Note 1 to entry: *Properties* describe the characteristics of a *Node*.

3.1.9

SourceNode

Node having a *Reference* to another *Node*

EXAMPLE: In the *Reference* "A contains B", "A" is the *SourceNode*.

3.1.10

TargetNode

Node that is referenced by another *Node*

EXAMPLE: In the *Reference* "A contains B", "B" is the *TargetNode*.

3.1.11

TypeDefinitionNode

Node that is used to define the type of another *Node*

Note 1 to entry: *ObjectType* and *VariableType Nodes* are *TypeDefinitionNodes*.

3.1.12

VariableType

Node that represents the type definition for a *Variable*

3.2 Abbreviated terms

UA	Unified Architecture
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

3.3 Conventions

3.3.1 Conventions for AddressSpace figures

Nodes and their *References* to each other are illustrated using figures. Figure 1 illustrates the conventions used in these figures.

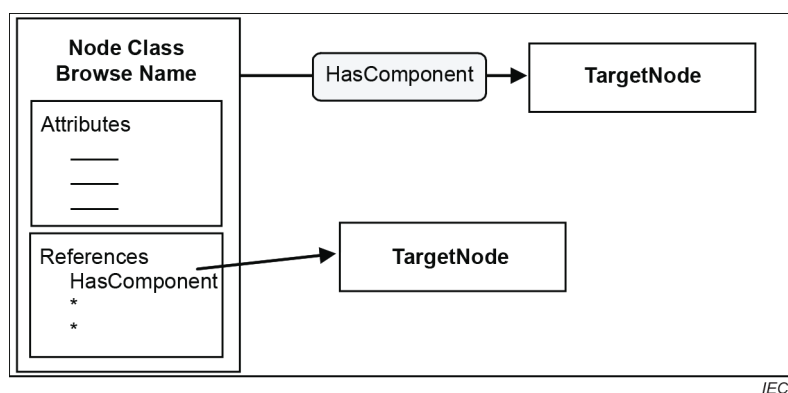


Figure 1 – AddressSpace Node diagrams

In these figures, rectangles represent *Nodes*. *Node* rectangles may be titled with one or two lines of text. When two lines are used, the first text line in the rectangle identifies the *NodeClass* and the second line contains the *BrowseName*. When one line is used, it contains the *BrowseName*.

Node rectangles may contain boxes used to define their *Attributes* and *References*. Specific names in these boxes identify specific *Attributes* and *References*.

Shaded rectangles with rounded corners and with arrows passing through them represent *References*. The arrow that passes through them begins at the *SourceNode* and points to the *TargetNode*. *References* may also be shown by drawing an arrow that starts at the *Reference* name in the "References" box and ends at the *TargetNode*.

3.3.2 Conventions for defining NodeClasses

Clause 5 defines *AddressSpace NodeClasses*. Table 1 describes the format of the tables used to define *NodeClasses*.

Table 1 – NodeClass Table Conventions

Name	Use	Data Type	Description
Attributes			
"Attribute name"	"M" or "O"	Data type of the <i>Attribute</i>	Defines the <i>Attribute</i>

Name	Use	Data Type	Description
References			
"Reference name"	"1", "0..1" or "0..*"	Not used	Describes the use of the <i>Reference</i> by the <i>NodeClass</i>
Standard Properties			
"Property name"	"M" or "O"	Data type of the <i>Property</i>	Defines the <i>Property</i>

The Name column contains the name of the *Attribute*, the name of the *ReferenceType* used to create a *Reference* or the name of a *Property* referenced using the *HasProperty Reference*.

The Use column defines whether the *Attribute* or *Property* is mandatory (M) or optional (O). When mandatory the *Attribute* or *Property* shall exist for every *Node* of the *NodeClass*. For *References* it specifies the cardinality. The following values may apply:

- "0..*" identifies that there are no restrictions, that is, the *Reference* does not have to be provided but there is no limitation how often it can be provided;
- "0..1" identifies that the *Reference* is provided at most once;
- "1" identifies that the *Reference* shall be provided exactly once.

The Data Type column contains the name of the *DataType* of the *Attribute* or *Property*. It is not used for *References*.

The Description column contains the description of the *Attribute*, the *Reference* or the *Property*.

Only this document may define *Attributes*. Thus, all *Attributes* of the *NodeClass* are specified in the table and may only be extended by other parts of the IEC 62541 series.

This document also defines *ReferenceTypes*, but *ReferenceTypes* may also be specified by a *Server* or by a *Client* using the *NodeManagement Services* specified in IEC 62541-4. Thus, the *NodeClass* tables contained in this document may contain the base *ReferenceType* called *References* identifying that any *ReferenceType* may be used for the *NodeClass*, including system specific *ReferenceTypes*. The *NodeClass* tables only specify how the *NodeClasses* can be used as *SourceNodes* of *References*, not as *TargetNodes*. If a *NodeClass* table allows a *ReferenceType* for its *NodeClass* to be used as *SourceNode*, this is also true for subtypes of the *ReferenceType*. However, subtypes of the *ReferenceType* may restrict its *SourceNodes*.

This document defines *Properties*, but *Properties* can be defined by other standards organizations or vendors and *Nodes* can have *Properties* that are not standardized. *Properties* defined in this document are defined by their name, which is mapped to the *BrowseName* having the *NamespaceIndex* 0, which represents the *Namespace* for OPC UA.

The Use column (optional or mandatory) does not imply a specific *ModellingRule* for *Properties*. Different *Server* implementations will choose to use *ModellingRules* appropriate for them.

4 AddressSpace concepts

4.1 Overview

Clause 4 defines the concepts of the *AddressSpace*. Clause 5 defines the *NodeClasses* of the *AddressSpace* representing the *AddressSpace* concepts. Clause 6 defines details on the type model for *ObjectTypes* and *VariableTypes*. Standard *ReferenceTypes*, *DataTypes* and *EventTypes* are defined in Clauses 7 to 9.

The informative Annex A describes general considerations on how to use the Address Space Model and the informative Annex B provides a UML Model of the Address Space Model. The normative Annex C defines a graphical notation for OPC UA data.

4.2 Object Model

The primary objective of the OPC UA *AddressSpace* is to provide a standard way for *Servers* to represent *Objects* to *Clients*. The OPC UA Object Model has been designed to meet this objective. It defines *Objects* in terms of *Variables* and *Methods*. It also allows relationships to other *Objects* to be expressed. Figure 2 illustrates the model.

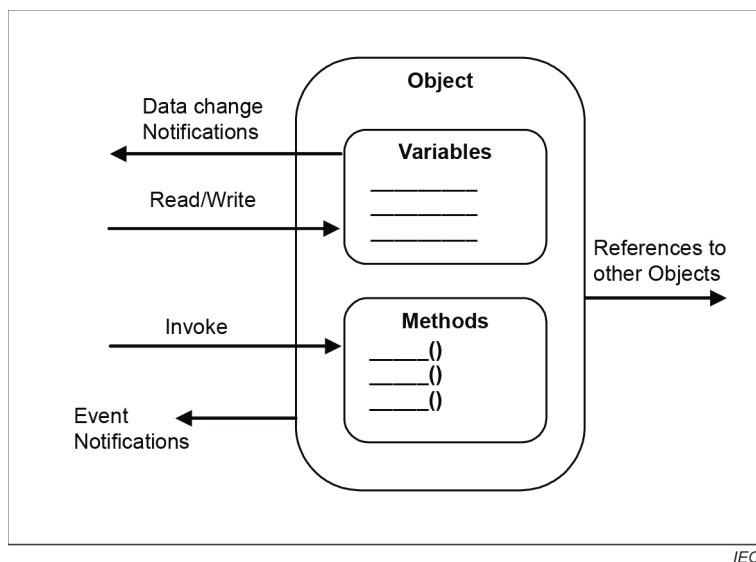


Figure 2 – OPC UA Object Model

The elements of this model are represented in the *AddressSpace* as *Nodes*. Each *Node* is assigned to a *NodeClass* and each *NodeClass* represents a different element of the Object Model. Clause 5 defines the *NodeClasses* used to represent this model.

4.3 Node Model

4.3.1 General

The set of *Objects* and related information that the OPC UA *Server* makes available to *Clients* is referred to as its *AddressSpace*. The model for *Objects* is defined by the OPC UA Object Model (see 4.2).

Objects and their components are represented in the *AddressSpace* as a set of *Nodes* described by *Attributes* and interconnected by *References*. Figure 3 illustrates the model of a *Node* and the remainder of 4.3 discusses the details of the Node Model.

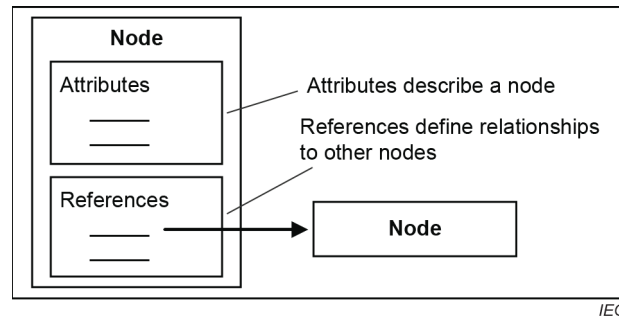


Figure 3 – AddressSpace Node Model

4.3.2 NodeClasses

NodeClasses are defined in terms of the *Attributes* and *References* that shall be instantiated (given values) when a *Node* is defined in the *AddressSpace*. *Attributes* are discussed in 4.3.3 and *References* in 4.3.4.

Clause 5 defines the *NodeClasses* for the OPC UA *AddressSpace*. These *NodeClasses* are referred to collectively as the metadata for the *AddressSpace*. Each *Node* in the *AddressSpace* is an instance of one of these *NodeClasses*. No other *NodeClasses* shall be used to define *Nodes*, and as a result, *Clients* and *Servers* are not allowed to define *NodeClasses* or extend the definitions of these *NodeClasses*.

4.3.3 Attributes

Attributes are data elements that describe *Nodes*. *Clients* can access *Attribute* values using Read, Write, Query, and Subscription/MonitoredItem *Services*. These *Services* are defined in IEC 62541-4.

Attributes are elementary components of *NodeClasses*. *Attribute* definitions are included as part of the *NodeClass* definitions in Clause 5 and, therefore, are not included in the *AddressSpace*.

Each *Attribute* definition consists of an attribute id (for attribute ids of *Attributes*, see IEC 62541-6), a name, a description, a data type and a mandatory/optional indicator. The set of *Attributes* defined for each *NodeClass* shall not be extended by *Clients* or *Servers*.

When a *Node* is instantiated in the *AddressSpace*, the values of the *NodeClass Attributes* are provided. The mandatory/optional indicator for the *Attribute* indicates whether the *Attribute* shall be instantiated.

4.3.4 References

References are used to relate *Nodes* to each other. They can be accessed using the browsing and querying *Services* defined in IEC 62541-4.

Like *Attributes*, they are defined as fundamental components of *Nodes*. Unlike *Attributes*, *References* are defined as instances of *ReferenceType Nodes*. *ReferenceType Nodes* are visible in the *AddressSpace* and are defined using the *ReferenceType NodeClass* (see 5.3).

The *Node* that contains the *Reference* is referred to as the *SourceNode* and the *Node* that is referenced is referred to as the *TargetNode*. The combination of the *SourceNode*, the *ReferenceType* and the *TargetNode* is used in OPC UA *Services* to uniquely identify *References*. Thus, each *Node* can reference another *Node* with the same *ReferenceType* only once. Any subtypes of concrete *ReferenceTypes* are considered to be equal to the base concrete *ReferenceTypes* when identifying *References* (see 5.3 for subtypes of *ReferenceTypes*). Figure 4 illustrates this model of a *Reference*.

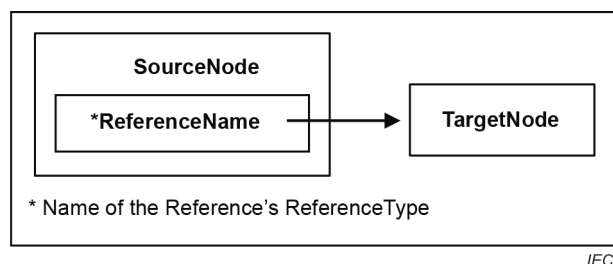


Figure 4 – Reference Model

The *TargetNode* of a *Reference* may be in the same *AddressSpace* or in the *AddressSpace* of another OPC UA *Server*. *TargetNodes* located in other *Servers* are identified in OPC UA *Services* using a combination of the remote *Server* name and the identifier assigned to the *Node* by the remote *Server*.

OPC UA does not require that *the TargetNode* exists, thus *References* may point to a *Node* that does not exist.

4.4 Variables

4.4.1 General

Variables are used to represent *values*. Two types of *Variables* are defined, *Properties* and *DataVariables*. They differ in the kind of data that they represent and whether they can contain other *Variables*.

4.4.2 Properties

Properties are *Server*-defined characteristics of *Objects*, *DataVariables* and other *Nodes*. *Properties* differ from *Attributes* in that they characterize *what* the *Node* represents, such as a device or a purchase order. *Attributes* define additional metadata that is instantiated for all *Nodes* from a *NodeClass*. *Attributes* are common to all *Nodes* of a *NodeClass* and only defined by this document whereas *Properties* can be *Server*-defined.

For example, an *Attribute* defines the *DataType* of *Variables* whereas a *Property* can be used to specify the engineering unit of some *Variables*.

To prevent recursion, *Properties* are not allowed to have *Properties* defined for them. To easily identify *Properties*, the *BrowseName* of a *Property* shall be unique in the context of the *Node* containing the *Properties* (see 5.6.3 for details).

A *Node* and its *Properties* shall always reside in the same *Server*.

4.4.3 DataVariables

DataVariables represent the content of an *Object*. For example, a file *Object* may be defined that contains a stream of bytes. The stream of bytes may be defined as a *DataVariable* that is an array of bytes. *Properties* may be used to expose the creation time and owner of the file *Object*.

For example, if a *DataVariable* is defined by a data structure that contains two fields, "startTime" and "endTime" then it might have a *Property* specific to that data structure, such as "earliestStartTime".

As another example, function blocks in control systems might be represented as *Objects*. The parameters of the function block, such as its setpoints, may be represented as *DataVariables*. The function block *Object* might also have *Properties* that describe its execution time and its type.

DataVariables may have additional *DataVariables*, but only if they are complex. In this case, their *DataVariables* shall always be elements of their complex definitions. Following the example introduced by the description of *Properties* in 4.4.2, the *Server* could expose "startTime" and "endTime" as separate components of the data structure.

As another example, a complex *DataVariable* may define an aggregate of temperature values generated by three separate temperature transmitters that are also visible in the *AddressSpace*. In this case, this complex *DataVariable* could define *HasComponent References* from it to the individual temperature values that it is composed of.

4.5 TypeDefinitionNodes

4.5.1 General

OPC UA Servers shall provide type definitions for *Objects* and *Variables*. The *HasTypeDefinition Reference* shall be used to link an instance with its type definition represented by a *TypeDefinitionNode*. Type definitions are required; however, IEC 62541-5 defines a *BaseObjectType*, a *PropertyType*, and a *BaseDataVariableType* so a *Server* can use such a base type if no more specialized type information is available. *Objects* and *Variables* inherit the *Attributes* specified by their *TypeDefinitionNode* (see 6.4 for details).

In some cases, the *NodeId* used by the *HasTypeDefinition Reference* will be well-known to *Clients* and *Servers*. Organizations may define *TypeDefinitionNodes* that are well-known in the industry. Well-known *NodeIds* of *TypeDefinitionNodes* provide for commonality across OPC UA Servers and allow *Clients* to interpret the *TypeDefinitionNode* without having to read it from the *Server*. Therefore, *Servers* may use well-known *NodeIds* without representing the corresponding *TypeDefinitionNodes* in their *AddressSpace*. However, the *TypeDefinitionNodes* shall be provided for generic *Clients*. These *TypeDefinitionNodes* may exist in another *Server*.

The following example, illustrated in Figure 5, describes the use of the *HasTypeDefinition Reference*. In this example, a setpoint parameter "SP" is represented as a *DataVariable* in the *AddressSpace*. This *DataVariable* is part of an *Object* not shown in the figure.

To provide for a common setpoint definition that can be used by other *Objects*, a specialized *VariableType* is used. Each setpoint *DataVariable* that uses this common definition will have a *HasTypeDefinition Reference* that identifies the common "SetPoint" *VariableType*.

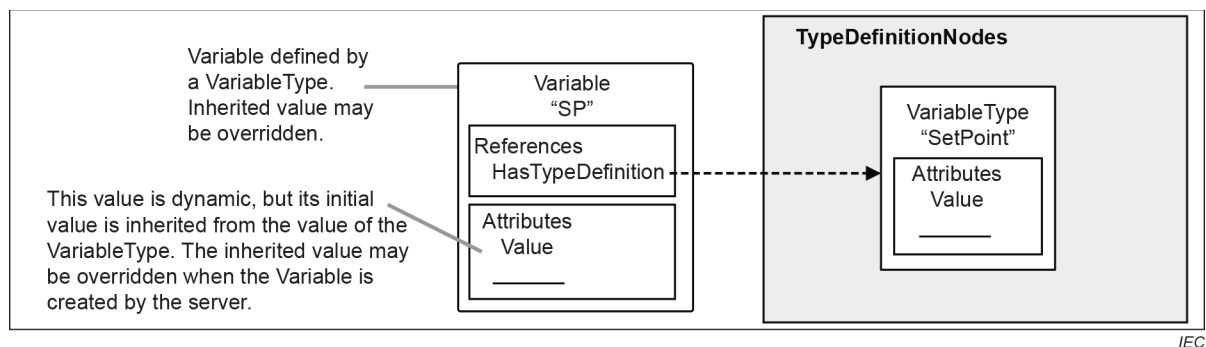


Figure 5 – Example of a Variable defined by a VariableType

4.5.2 Complex TypeDefinitionNodes and their InstanceDeclarations

TypeDefinitionNodes can be complex. A complex *TypeDefinitionNode* also defines *References* to other *Nodes* as part of the type definition. The *ModellingRules* defined in 6.4.4 specify how those *Nodes* are handled when creating an instance of the type definition.

A *TypeDefinitionNode* references instances instead of other *TypeDefinitionNodes* to allow unique names for several instances of the same type, to define default values and to add *References* for those instances that are specific to this complex *TypeDefinitionNode* and not to the *TypeDefinitionNode* of the instance. For example, in Figure 6 the *ObjectType* "AI_BLK_TYPE", representing a function block, has a *HasComponent Reference* to a *Variable* "SP" of the *VariableType* "SetPoint". "AI_BLK_TYPE" could have an additional setpoint *Variable* of the same type using a different name. It could add a *Property* to the *Variable* that was not defined by its

TypeDefinitionNode "SetPoint". And it could define a default value for "SP", that is, each instance of "AI_BLK_TYPE" would have a *Variable* "SP" initially set to this value.

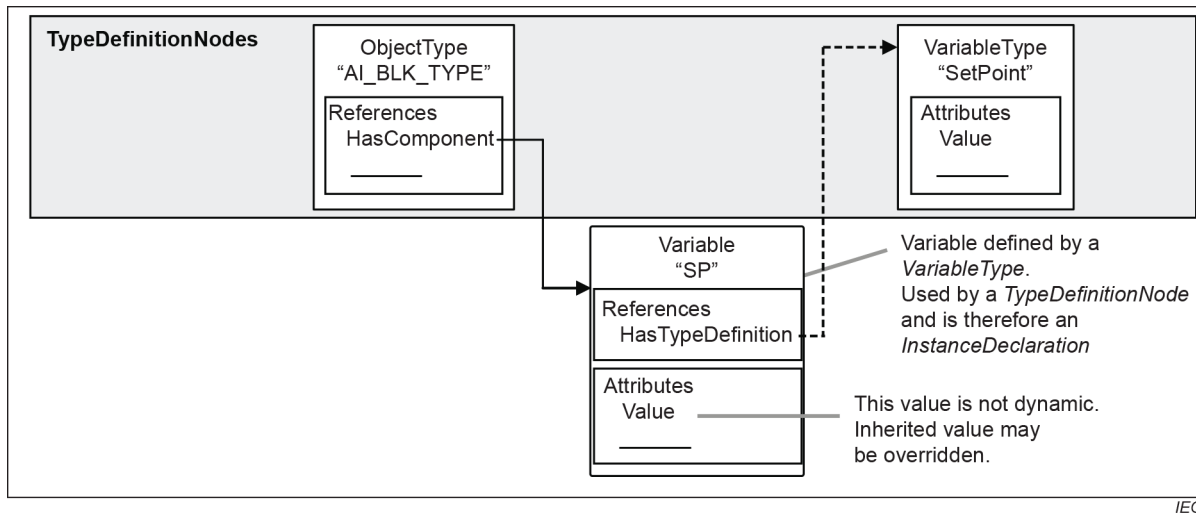


Figure 6 – Example of a Complex TypeDefinition

This approach is commonly used in object-oriented programming languages in which the variables of a class are defined as instances of other classes. When the class is instantiated, each variable is also instantiated, but with the default values (constructor values) defined for the containing class. That is, typically, the constructor for the component class runs first, followed by the constructor for the containing class. The constructor for the containing class may override component values set by the component class.

To distinguish instances used for the type definitions from instances that represent real data, those instances are called *InstanceDeclarations*. However, this term is used to simplify this specification, if an instance is an *InstanceDeclaration* or not is only visible in the *AddressSpace* by following its *References*. Some instances may be shared and therefore referenced by *TypeDefinitionNodes*, *InstanceDeclarations* and instances. This is similar to class variables in object-oriented programming languages.

4.5.3 Subtyping

This document allows subtyping of type definitions. The subtyping rules are defined in Clause 6. Subtyping of *ObjectTypes* and *VariableTypes* allows:

- *Clients* that only know the supertype to handle an instance of the subtype as if it were an instance of the supertype;
- instances of the supertype to be replaced by instances of the subtype;
- specialized types that inherit common characteristics of the base type.

In other words, subtypes reflect the structure defined by their supertype but may add additional characteristics. For example, a vendor may wish to extend a general "TemperatureSensor" *VariableType* by adding a *Property* providing the next maintenance interval. The vendor would do this by creating a new *VariableType* which is a *TargetNode* for a *HasSubtype* reference from the original *VariableType* and adding the new *Property* to it.

4.5.4 Instantiation of complex TypeDefinitionNodes

The instantiation of complex *TypeDefinitionNodes* depends on the *ModellingRules* defined in 6.4.4. However, the intention is that instances of a type definition will reflect the structure defined by the *TypeDefinitionNode*. Figure 7 shows an instance of the *TypeDefinitionNode* "AI_BLK_TYPE", where the *ModellingRule Mandatory*, defined in 6.4.4.5.2, was applied for its containing *Variable*. Thus, an instance of "AI_BLK_TYPE", called "AI_BLK_1", has a *HasTypeDefinition Reference* to "AI_BLK_TYPE". It also contains a *Variable* "SP" having the same *BrowseName* as the *Variable* "SP" used by the *TypeDefinitionNode* and thereby reflects the structure defined by the *TypeDefinitionNode*.

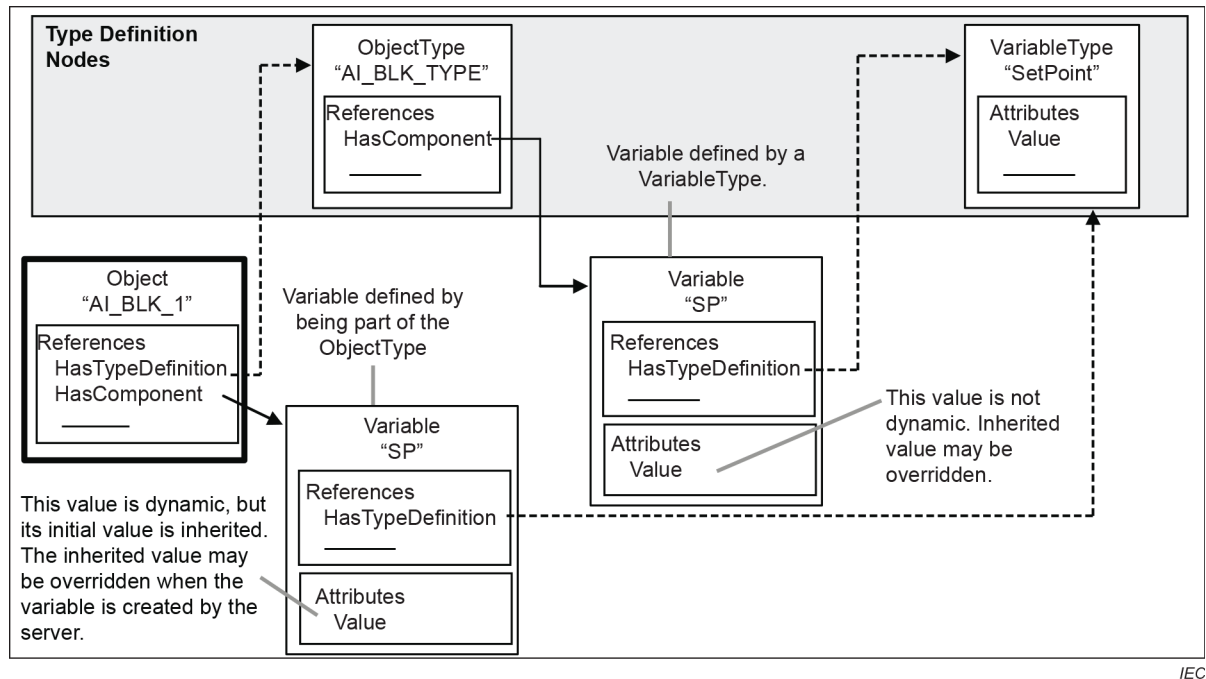


Figure 7 – Object and its Components defined by an ObjectType

A *Client* knowing the *ObjectType* "AI_BLK_TYPE" can use this knowledge to directly browse to the containing *Nodes* for each instance of this type. This allows programming against the *TypeDefinitionNode*. For example, a graphical element may be programmed in the *Client* that handles all instances of "AI_BLK_TYPE" in the same way by showing the value of "SP".

There are several constraints related to programming against the *TypeDefinitionNode*. A *TypeDefinitionNode* or an *InstanceDeclaration* shall never reference two *Nodes* having the same *BrowseName* using forward *Hierarchical References*. Instances based on *InstanceDeclarations* shall always keep the same *BrowseName* as the *InstanceDeclaration* they are derived from. A special *Service* defined in IEC 62541-4 called *TranslateBrowsePathsToNodeIds* may be used to identify the instances based on the *InstanceDeclarations*. Using the simple *Browse Service* might not be sufficient since the uniqueness of the *BrowseName* is only required for *TypeDefinitionNodes* and *InstanceDeclarations*, not for other instances. Thus, "AI_BLK_1" may have another *Variable* with the *BrowseName* "SP", although this one would not be derived from an *InstanceDeclaration* of the *TypeDefinitionNode*.

Instances derived from an *InstanceDeclaration* shall be of the same *TypeDefinitionNode* or a subtype of this *TypeDefinitionNode*.

A *TypeDefinitionNode* and its *InstanceDeclarations* shall always reside in the same *Server*. However, instances may point with their *HasTypeDefinition Reference* to a *TypeDefinitionNode* in a different *Server*.

4.6 Event Model

4.6.1 General

The Event Model defines a general purpose eventing system that can be used in many diverse vertical markets.

Events represent specific transient occurrences. System configuration changes and system errors are examples of *Events*. *Event Notifications* report the occurrence of an *Event*. *Events* defined in this document are not directly visible in the OPC UA *AddressSpace*. *Objects* and *Views* can be used to subscribe to *Events*. The *EventNotifier Attribute* of those *Nodes* identifies if the *Node* allows subscribing to *Events*. *Clients* subscribe to such *Nodes* to receive *Notifications* of *Event* occurrences.

Event Subscriptions use the Subscription/MonitoredItem Services defined in IEC 62541-4 to subscribe to the *Event Notifications* of a *Node*.

Any OPC UA Server that supports eventing shall expose at least one *Node* as *EventNotifier*. The *Server Object* defined in IEC 62541-5 is used for this purpose. *Events* generated by the *Server* are available via this *Server Object*. A *Server* is not expected to produce *Events* if the connection to the event source is down for some reason (i.e. the system is offline).

Events may also be exposed through other *Nodes* anywhere in the *AddressSpace*. These *Nodes* (identified via the *EventNotifier Attribute*) provide some subset of the *Events* generated by the *Server*. The position in the *AddressSpace* dictates what this subset will be. For example, a process area *Object* representing a functional area of the process would provide *Events* originating from that area of the process only. It should be noted that this is only an example and it is fully up to the *Server* to determine what *Events* should be provided by which *Node*.

4.6.2 EventTypes

Each *Event* is of a specific *EventType*. A *Server* may support many types. This document defines the *BaseEventType* that all other *EventTypes* derive from. It is expected that other companion specifications will define additional *EventTypes* deriving from the base types defined in this document.

The *EventTypes* supported by a *Server* are exposed in the *AddressSpace* of a *Server*. *EventTypes* are represented as *ObjectTypes* in the *AddressSpace* and do not have a special *NodeClass* associated to them. IEC 62541-5 defines how a *Server* exposes the *EventTypes* in detail.

EventTypes defined in this document are specified as abstract and therefore never instantiated in the *AddressSpace*. Event occurrences of those *EventTypes* are only exposed via a *Subscription*. *EventTypes* exist in the *AddressSpace* to allow *Clients* to discover the *EventType*. This information is used by a *Client* when establishing and working with *Event Subscriptions*. *EventTypes* defined by other parts of IEC 62541 or companion specifications as well as *Server* specific *EventTypes* may be defined as not abstract; therefore, instances of those *EventTypes* may be visible in the *AddressSpace* although *Events* of those *EventTypes* are also accessible via the *Event Notification* mechanisms.

Standard *EventTypes* are described in Clause 9. Their representation in the *AddressSpace* is specified in IEC 62541-5.

4.6.3 Event Categorization

Events can be categorized by creating new *EventTypes* which are subtypes of existing *EventTypes* but do not extend an existing type. They are used only to identify an event as being of the new *EventType*. For example, the *EventType* *DeviceFailureEventType* could be subtyped into *TransmitterFailureEventType* and *ComputerFailureEventType*. These new subtypes would not add new *Properties* or change the semantic inherited from the *DeviceFailureEventType* other than purely for categorization of the *Events*.

Event sources can also be organized into groups by using the *Event ReferenceTypes* described in 7.17 and 7.18. For example, a *Server* may define *Objects* in the *AddressSpace* representing *Events* related to physical devices, or *Event* areas of a plant or functionality contained in the *Server*. *Event References* would be used to indicate which *Event* sources represent physical devices and which ones represent some *Server*-based functionality. In addition, *References* can be used to group the physical devices or *Server*-based functionality into hierarchical *Event* areas. In some cases, an *Event* source may be categorized as being both a device and a *Server* function. In this case, two relationships would be established. Refer to the description of the *Event ReferenceTypes* for additional examples.

Clients can select a category or categories of *Events* by defining content filters that include terms specifying the *EventType* of the *Event* or a grouping of *Event* sources. The two mechanisms allow for a single *Event* to be categorized in multiple manners. A *Client* could obtain all *Events* related to a physical device or all failures of a particular device.

4.7 Methods

Methods are “lightweight” functions, whose scope is bounded by an owning (see Note) *Object*, similar to the methods of a class in object-oriented programming or an owning *ObjectType*, similar to static methods of a class. *Methods* are invoked by a *Client*, proceed to completion on the *Server* and return the result to the *Client*. The lifetime of the *Method*’s invocation instance begins when the *Client* calls the *Method* and ends when the result is returned.

NOTE The owning *Object* or *ObjectType* is specified in the service call when invoking the *Method*.

While *Methods* may affect the state of the owning *Object*, they have no explicit state of their own. In this sense, they are stateless. *Methods* can have a varying number of input arguments and return resultant arguments. Each *Method* is described by a *Node* of the *Method NodeClass*. This *Node* contains the metadata that identifies the *Method*’s arguments and describes its behaviour.

Methods are invoked by using the Call *Service* defined in IEC 62541-4.

Clients discover the *Methods* supported by a *Server* by browsing for the owning *Objects References* that identify their supported *Methods*.

4.8 Roles

4.8.1 Overview

A *Role* is a function assumed by a *Client* when it accesses a *Server*. *Roles* are used to separate authentication (determining who a *Client* is) from authorization (determining what the *Client* is allowed to do). By separating these tasks *Servers* can allow centralized services to manage user identities and credentials while the *Server* only manages the *Permissions* on its *Nodes* assigned to *Roles*.

The set of *Roles* supported by a *Server* are published as components of the *Roles Object* defined in IEC 62541-5. *Servers* should define a base set of *Roles* and allow configuration *Clients* to add system specific *Roles*.

When a *Session* is created, the *Server* shall determine what *Roles* are granted to that *Session*. This document defines standard mapping rules which *Servers* may support. *Servers* may also use vendor specific mapping rules in addition to or instead of the standard rules.

The standard mapping rules allow *Roles* to be granted based on:

- user identity;
- application identity;
- endpoint.

User identity mappings can be based on user names, user certificates or user groups. Well known groups include ‘AuthenticatedUser’ (any user with valid credentials) and ‘Anonymous’ (no user credentials provided).

Application identity mappings are based on the *ApplicationUri* specified in the *Client Certificate*. Application identity can only be enforced if the *Client* proves possession of a trusted *Certificate* by using it to create a *Secure Channel* or by providing a signature in *ActivateSession* (see IEC 62541-4).

Endpoint identity mappings are based on the URL used to connect to the *Server*. Endpoint identity can be used to restrict access to *Clients* running on particular networks.

IEC 62541-5 defines the *Objects*, *Methods* and *DataTypes* used to represent and manage these mapping rules in the *AddressSpace*.

4.8.2 Well-known Roles

All *Servers* should support the well-known *Roles* which are defined in Table 2. The *NodeIds* for the well-known *Roles* are defined in IEC 62541-6.

Table 2 – Well-known Roles

BrowseName	Suggested Permissions
Anonymous	The <i>Role</i> has very limited access for use when a <i>Session</i> has anonymous credentials.
AuthenticatedUser	The <i>Role</i> has limited access for use when a <i>Session</i> has valid non-anonymous credentials but has not been explicitly granted access to a <i>Role</i> .
Observer	The <i>Role</i> is allowed to browse, read live data, read historical data/events or subscribe to data/events.
Operator	The <i>Role</i> is allowed to browse, read live data, read historical data/events or subscribe to data/events. In addition, the <i>Session</i> is allowed to write some live data and call some <i>Methods</i> .
Engineer	The <i>Role</i> is allowed to browse, read/write configuration data, read historical data/events, call <i>Methods</i> or subscribe to data/events.
Supervisor	The <i>Role</i> is allowed to browse, read live data, read historical data/events, call <i>Methods</i> or subscribe to data/events.
ConfigureAdmin	The <i>Role</i> is allowed to change the non-security related configuration settings.
SecurityAdmin	The <i>Role</i> is allowed to change security related settings.

4.8.3 Evaluating Permissions with Roles

When a *Client* attempts to access a *Node*, the *Server* goes through the list of *Roles* granted to the *Session* and logically ORs the *Permissions* for the *Role* on the *Node*. If there are no *Node specific Permissions* then the default *Permissions* for the *Role* in the *DefaultRolePermissions Property* of the *NamespaceMetadata* for the namespace the *Node* belongs to are used (see IEC 62541-5). The resulting mask is the effective *Permissions*. If the bits corresponding to current operation are set, then the operation can proceed. If they are not set the *Server* returns *Bad_UserAccessDenied*.

Roles appear under the *Roles Object* in the *Server AddressSpace*. Each *Role* has mapping rules defined which appear as *Properties* of the *Role Object* (see IEC 62541-5). The examples shown in Table 3 illustrate how the standard mapping rules can be used to determine which *Roles* a *Session* has access to and, consequently, the *Permissions* that are granted to the *Session*.

Table 3 – Example Roles

Role	Mapping Rules	Description
Anonymous	Identities = Anonymous Applications = Endpoints =	An identity mapping rule that specifies the <i>Role</i> applies to anonymous users.
AuthenticatedUser	Identities = AuthenticatedUser Applications = Endpoints =	An identity mapping rule that specifies the <i>Role</i> applies to authenticated users.
Operator1	Identities = User with name 'Joe' Applications = urn:OperatorStation1 Endpoints =	An identity mapping rule that specifies specific users that have access to the <i>Role</i> with an application rule that restricts access to a single Client application.
Operator2	Identities = Users with name 'Joe' or 'Ann' Applications = urn:OperatorStation2 Endpoints =	An identity mapping rule that specifies specific users that have access to the <i>Role</i> with an application rule that restricts access to a single Client application.
Supervisor	Identities = User with name 'Root' Applications = Endpoints =	An identity mapping rule that specifies specific users that have access to the <i>Role</i>
Administrator	Identities = User with name 'Root' Applications = Endpoints = opc.tcp://127.0.0.1:48000	An identity mapping rule that specifies specific users that have access to the <i>Role</i> when they connect via a specific Endpoint.

The examples also make use of the *Nodes* defined in Table 4. The table specifies the value of the *RolePermissions Attribute* for each *Node*.

Table 4 – Example Nodes

Node	Role Permissions
Unit1.Measurement	AuthenticatedUser = Browse Operator1 = Browse, Read
Unit2.Measurement	AuthenticatedUser = Browse Operator2 = Browse, Read
SetPoint	AuthenticatedUser = Browse Operator1 and Operator2 = Browse, Read, Write Supervisor = Browse, Read
DisableDevice	AuthenticatedUser = Browse Operator1 and Operator2 = Browse, Read Administrator = Browse, Read, Write

When a *Client* creates a *Session* the *Roles* assigned to the *Session* depend on the rules defined for each *Role*. Table 5 lists the assigned *Roles* for different *Sessions* created with different *Users*, *Client* applications and endpoints.

Table 5 – Example Role assignment

User Provided by Client	Roles Assigned to Session
Anonymous	Anonymous
Sam	AuthenticatedUser
Joe using OperatorStation1 application.	AuthenticatedUser, Operator1
Joe using OperatorStation2 application.	AuthenticatedUser, Operator2
Joe using generic application.	AuthenticatedUser
Root using OperatorStation1 application.	AuthenticatedUser, Supervisor
Root using generic application and 127.0.0.1 endpoint.	AuthenticatedUser, Supervisor, Administrator
Root using generic application and another endpoint.	AuthenticatedUser, Supervisor

When a *Client* application accesses a *Node* the *RolePermissions* for the *Node* are compared to the *Roles* assigned to the *Session*. Any *Permission* available to at least one *Role* is granted to the *Client*. Table 6 provides a number of scenarios and examples and the resulting decision on access.

Table 6 – Examples of evaluating access

Use Case	Role Permissions
Anonymous user on localhost browses Unit1.Measurement <i>Node</i> .	Access denied because no rule defined for Anonymous users.
User 'Sam' using OperatorStation1 application browses Unit1.Measurement <i>Node</i> .	Allowed because AuthenticatedUser is granted Browse <i>Permission</i> .
User 'Sam' using OperatorStation2 application reads <i>Value</i> of Unit1.Measurement <i>Node</i> .	Access denied because AuthenticatedUser is not granted Read <i>Permission</i> .
User 'Joe' using OperatorStation1 application reads <i>Value</i> of Unit1.Measurement <i>Node</i> .	Allowed because Operator1 is granted Read <i>Permission</i> .
User 'Joe' using OperatorStation2 application reads <i>Value</i> of Unit1.Measurement <i>Node</i> .	Access denied because AuthenticatedUser and Operator2 are not granted Read <i>Permission</i> .
User 'Joe' using generic OPC UA application reads <i>Value</i> of Measurement <i>Node</i> .	Access denied because AuthenticatedUser is not granted Read <i>Permission</i> .
User 'Joe' using OperatorStation1 application writes <i>Value</i> of SetPoint <i>Node</i> .	Allowed because Operator1 is granted Write <i>Permission</i> .
User 'Root' using OperatorStation1 application writes the <i>Value</i> of SetPoint <i>Node</i> .	Denied because AuthenticatedUser and Supervisor are not granted Write <i>Permission</i> .
User 'Joe' using OperatorStation1 application writes <i>Value</i> of DisableDevice <i>Node</i> .	Access denied because AuthenticatedUser and Operator1 are not granted Write <i>Permission</i> .
User 'Root' using OperatorStation1 application writes the <i>Value</i> of DisableDevice <i>Node</i> .	Access denied because AuthenticatedUser and Supervisor are not granted Write <i>Permission</i> .
User 'Root' using endpoint 127.0.0.1 to write <i>Value</i> of DisableDevice <i>Node</i> .	Allowed because Administrator is granted Write <i>Permission</i> .

5 Standard NodeClasses

5.1 Overview

Clause 5 defines the *NodeClasses* used to define *Nodes* in the OPC UA *AddressSpace*. *NodeClasses* are derived from a common *Base NodeClass*. This *NodeClass* is defined first, followed by those used to organize the *AddressSpace* and then by the *NodeClasses* used to represent *Objects*.

The *NodeClasses* defined to represent *Objects* fall into three categories: those used to define instances, those used to define types for those instances and those used to define data types. Subclause 6.3 describes the rules for subtyping and 6.4 the rules for instantiation of the type definitions.

5.2 Base NodeClass

5.2.1 General

The OPC UA Address Space Model defines a *Base NodeClass* from which all other *NodeClasses* are derived. The derived *NodeClasses* represent the various components of the OPC UA Object Model (see 4.2). The *Attributes* of the *Base NodeClass* are specified in Table 7. There are no *References* specified for the *Base NodeClass*.

Table 7 – Base NodeClass

Name	Use	Data Type	Description
Attributes			
NodeId	M	NodeId	See 5.2.2
NodeClass	M	NodeClass	See 5.2.3
BrowseName	M	QualifiedName	See 5.2.4
DisplayName	M	LocalizedText	See 5.2.5
Description	O	LocalizedText	See 5.2.6
WriteMask	O	AttributeWriteMask	See 5.2.7
UserWriteMask	O	AttributeWriteMask	See 5.2.8
RolePermissions	O	RolePermissionType[]	See 5.2.9
UserRolePermissions	O	RolePermissionType[]	See 5.2.10
AccessRestrictions	O	AccessRestrictionsType	See 5.2.11
References			
			No <i>References</i> specified for this <i>NodeClass</i>

5.2.2 NodeId

Nodes are unambiguously identified using a constructed identifier called the *NodeId*. Some *Servers* may accept alternative *NodeIds* in addition to the canonical *NodeId* represented in this *Attribute*. A *Server* shall persist the *NodeId* of a *Node*, that is, it shall not generate new *NodeIds* when rebooting. The structure of the *NodeId* is defined in 8.2.

5.2.3 NodeClass

The *NodeClass Attribute* identifies the *NodeClass* of a *Node*. Its data type is defined in 8.30.

5.2.4 BrowseName

Nodes have a *BrowseName Attribute* that is used as a non-localized human-readable name when browsing the *AddressSpace* to create paths out of *BrowseNames*. The *TranslateBrowsePathsToNodeIds Service* defined in IEC 62541-4 can be used to follow a path constructed of *BrowseNames*.

A *BrowseName* should never be used to display the name of a *Node*. The *DisplayName* should be used instead for this purpose.

Unlike *NodeIds*, the *BrowseName* cannot be used to unambiguously identify a *Node*. Different *Nodes* may have the same *BrowseName*.

Subclause 8.3 defines the structure of the *BrowseName*. It contains a namespace and a string. The namespace is provided to make the *BrowseName* unique in some cases in the context of a *Node* (e.g. *Properties* of a *Node*) although not unique in the context of the *Server*. If different organizations define *BrowseNames* for *Properties*, the namespace of the *BrowseName* provided by the organization makes the *BrowseName* unique, although different organizations may use the same string having a slightly different meaning.

Servers may often choose to use the same namespace for the *NodeId* and the *BrowseName*. However, if they want to provide a standard *Property*, its *BrowseName* shall have the namespace of the standards body although the namespace of the *NodeId* reflects something else, for example the local *Server*.

It is recommended that standards bodies defining standard type definitions use their namespace for the *NodeId* of the *TypeDefinitionNode* as well as for the *BrowseName* of the *TypeDefinitionNode*.

The string-part of the *BrowseName* is case sensitive. That is, *Clients* shall consider them case sensitive. *Servers* are allowed to handle *BrowseNames* passed in *Service* requests as case insensitive. Examples are the *TranslateBrowsePathsToNodeIds Service* or *Event* filter.

5.2.5 DisplayName

The *DisplayName Attribute* contains the localized name of the *Node*. *Clients* should use this *Attribute* if they want to display the name of the *Node* to the user. They should not use the *BrowseName* for this purpose. The *Server* may maintain one or more localized representations for each *DisplayName*. *Clients* negotiate the locale to be returned when they open a session with the *Server*. Refer to IEC 62541-4 for a description of session establishment and locales. Subclause 8.5 defines the structure of the *DisplayName*. The string part of the *DisplayName* is restricted to 512 characters.

5.2.6 Description

The optional *Description Attribute* shall explain the meaning of the *Node* in a localized text using the same mechanisms for localization as described for the *DisplayName* in 5.2.5.

5.2.7 WriteMask

The optional *WriteMask Attribute* exposes the possibilities of a *Client* to write the *Attributes* of the *Node*. The *WriteMask Attribute* does not take any user access rights into account, that is, although an *Attribute* is writable this may be restricted to a certain user/user group.

If the OPC UA *Server* does not have the ability to get the *WriteMask* information for a specific *Attribute* from the underlying system, it should state that it is writable. If a write operation is called on the *Attribute*, the *Server* should transfer this request and return the corresponding *StatusCode* if such a request is rejected. *StatusCodes* are defined in IEC 62541-4.

The *AttributeWriteMask DataType* is defined in 8.60.

5.2.8 UserWriteMask

The optional *UserWriteMask Attribute* exposes the possibilities of a *Client* to write the *Attributes* of the *Node* taking user access rights into account. It uses the *AttributeWriteMask DataType* which is defined in 8.60.

The *UserWriteMask Attribute* can only further restrict the *WriteMask Attribute*, when it is set to not writable in the general case that applies for every user.

Clients cannot assume an *Attribute* can be written based on the *UserWriteMask 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.

5.2.9 RolePermissions

The optional *RolePermissions Attribute* specifies the *Permissions* that apply to a *Node* for all *Roles* which have access to the *Node*. The value of the *Attribute* is an array of *RolePermissionType Structures* (see Table 8).

Table 8 – RolePermissionType

Name	Type	Description
RolePermissionType	Structure	Specifies the <i>Permissions</i> for a <i>Role</i>
roleId	NodeId	The <i>NodeId</i> of the <i>Role Object</i> .
permissions	PermissionType	A mask specifying which <i>Permissions</i> are available to the <i>Role</i> .

Servers may allow administrators to write to the *RolePermissions Attribute*.

If not specified, the value of *DefaultRolePermissions Property* from the *NamespaceMetadata Object* associated with the *Node* shall be used instead. If the *NamespaceMetadata Object* does not define the *Property* or does not exist, then the *Server* should not publish any information about how it manages *Permissions*.

If a *Server* supports *Permissions* for a particular *Namespace* it shall add the *DefaultRolePermissions Property* to the *NamespaceMetadata Object* for that *Namespace* (see Figure 8). If a particular *Node* in the *Namespace* needs to override the default values, the *Server* adds the *RolePermissions Attribute* to the *Node*. The *DefaultRolePermissions Property* and *RolePermissions Attribute* shall only be readable by administrators. If a *Server* allows the *Permissions* to be changed these values shall be writable. If the *Server* allows the *Permissions* to be overridden for a particular *Node* but does not currently have any *Node Permissions* configured, then the value of the *Attribute* shall be an empty array. If the administrator wishes to remove overridden *Permissions*, an empty array shall be written to this *Attribute*. *Servers* shall prevent *Permissions* from being changed in such a way as to render the *Server* inoperable.

If a *Server* publishes information about the *Roles* for a *Namespace* assigned to the current *Session*, it shall add the *DefaultUserRolePermissions Property* to the *NamespaceMetadata Object* for that *Namespace*. The value of this *Property* shall be a readonly list of *Permissions* for each *Role* assigned to the current *Session*. If a particular *Node* in the *Namespace* overrides the default *RolePermissions* the *Server* shall also override the *DefaultUserRolePermissions* by adding the *UserRolePermissions Attribute* to the *Node*. If the *Server* allows the *Permissions* to be overridden for a particular *Node* but does not currently have any *Node Permissions* configured, then the *Server* shall return the value of the *DefaultUserRolePermissions Property* for the *Node Namespace*.

If a *Server* implements a vendor specific *Role Permission* model for a *Namespace*, it shall not add the *DefaultRolePermissions* or *DefaultUserRolePermissions Properties* to the *Namespace Metadata Object*.

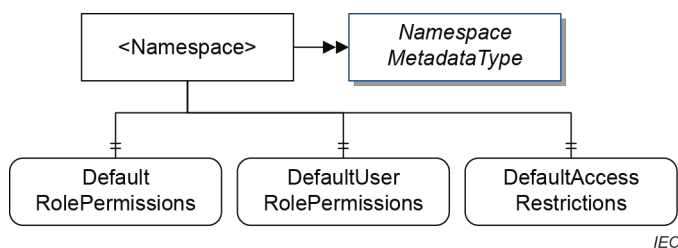


Figure 8 – Permissions in the Address Space

5.2.10 UserRolePermissions

The optional *UserRolePermissions Attribute* specifies the *Permissions* that apply to a *Node* for all *Roles* granted to current *Session*. The value of the *Attribute* is an array of *RolePermissionType Structures* (see Table 8).

Clients may determine their effective *Permissions* by performing a logical OR of *Permissions* for each *Role* in the array.

The value of this *Attribute* is derived from the rules used by the *Server* to map *Sessions* to *Roles*. This mapping may be vendor specific or it may use the standard *Role* model defined in 4.8.

This *Attribute* shall not be writable.

If not specified, the value of *DefaultUserRolePermissions Property* from the *Namespace Metadata Object* associated with the *Node* is used instead. If the *NamespaceMetadata Object* does not define the *Property* or does not exist, then the *Server* does not publish any information about *Roles* mapped to the current *Session*.

5.2.11 AccessRestrictions

The optional *AccessRestrictions Attribute* specifies the *AccessRestrictions* that apply to a *Node*. Its data type is defined in 8.56. If a *Server* supports *AccessRestrictions* for a particular *Namespace* it adds the *DefaultAccessRestrictions Property* to the *NamespaceMetadata Object* for that *Namespace* (see Figure 8). If a particular *Node* in the *Namespace* needs to override the default value the *Server* adds the *AccessRestrictions Attribute* to the *Node*.

If a *Server* implements a vendor specific access restriction model for a *Namespace*, it does not add the *DefaultAccessRestrictions Property* to the *NamespaceMetadata Object*.

5.3 ReferenceType NodeClass

5.3.1 General

References are defined as instances of *ReferenceType Nodes*. *ReferenceType Nodes* are visible in the *AddressSpace* and are defined using the *ReferenceType NodeClass* as specified in Table 9. In contrast, a *Reference* is an inherent part of a *Node* and no *NodeClass* is used to represent *References*.

This document defines a set of *ReferenceTypes* provided as an inherent part of the OPC UA Address Space Model. These *ReferenceTypes* are defined in Clause 7 and their representation in the *AddressSpace* is defined in IEC 62541-5. *Servers* may also define *ReferenceTypes*. In addition, IEC 62541-4 defines *NodeManagement Services* that allow *Clients* to add *ReferenceTypes* to the *AddressSpace*.