DIN EN IEC 62541-6

DIN

ICS 35.200; 35.240.50

Entwurf

Einsprüche bis 2019-10-09
Vorgesehen als Ersatz für
DIN EN 62541-6:2015-11

OPC Unified Architecture – Teil 6: Protokollabbildungen (IEC 65E/607/CDV:2018); Englische Fassung prEN IEC 62541-6:2018

OPC unified architecture –
Part 6: Mappings
(IEC 65E/607/CDV:2018);
English version prEN IEC 62541-6:2018

Architecture unifiée OPC –
Partie 6: Correspondances (Mappings)
(IEC 65E/607/CDV:2018);
Version anglaise prEN IEC 62541-6:2018

Anwendungswarnvermerk

Dieser Norm-Entwurf mit Erscheinungsdatum 2019-08-09 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

Weil die beabsichtigte Norm von der vorliegenden Fassung abweichen kann, ist die Anwendung dieses Entwurfs besonders zu vereinbaren.

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal von DIN unter www.din.de/go/entwuerfe bzw. für Norm-Entwürfe der DKE auch im Norm-Entwurfs-Portal der DKE unter www.entwuerfe.normenbibliothek.de, sofern dort wiedergegeben;
- oder als Datei per E-Mail an dke@vde.com möglichst in Form einer Tabelle. Die Vorlage dieser Tabelle kann im Internet unter www.din.de/go/stellungnahmen-norm-entwuerfe oder für Stellungnahmen zu Norm-Entwürfen der DKE unter www.dke.de/stellungnahme abgerufen werden;
- oder in Papierform an die DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE, Stresemannallee 15, 60596 Frankfurt am Main.

Die Empfänger dieses Norm-Entwurfs werden gebeten, mit ihren Kommentaren jegliche relevanten Patentrechte, die sie kennen, mitzuteilen und unterstützende Dokumentationen zur Verfügung zu stellen.

Gesamtumfang 106 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

AVA Documents Document Sharing Hub

Full document details: E DIN EN IEC 62541-6:2019-09

Nationales Vorwort

Die Englische Fassung des europäischen Dokuments prEN IEC 62541-6:2018 „OPC Unified Architecture – Teil 6: Protokollabbildungen“ (Entwurf in der Umfrage) ist unverändert in diesen Norm-Entwurf übernommen worden.

Die Internationale Elektrotechnische Kommission (IEC) und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dem entsprechend ist das internationale Dokument IEC 65E/607/CDV:2018 „OPC unified architecture – Part 6: Mappings“ unverändert in den Entwurf prEN IEC 62541-6:2018 übernommen worden.

Das internationale Dokument wurde vom SC 65E „Devices and integration in enterprise systems“ der Internationalen Elektrotechnischen Kommission (IEC) erarbeitet.

Bei der Abstimmung zu dem Europäischen Schluss-Entwurf bei CENELEC und dem Internationalen Schluss-Entwurf bei IEC [Final Draft International Standard (FDIS)] sind jeweils nur „JA/NEIN“-Entscheidungen möglich, wobei „NEIN“-Entscheidungen fundiert begründet werden müssen. Dokumente, die bei CENELEC als Europäische Norm angenommen und ratifiziert werden, sind unverändert als Deutsche Normen zu übernehmen.

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

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 den vorliegenden Norm-Entwurf erfüllt.

Da sich die Benutzer des vorliegenden Norm-Entwurfs der englischen Sprache als Fachsprache bedienen, wird die englische Fassung der prEN IEC 62541-6 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt, und Unterabschnitt 3.1, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Norm-Entwurfs hinzugefügt. Für einige der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Dieser Norm-Entwurf steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

Es ist vorgesehen, auch bei der entsprechenden zukünftigen Deutschen Norm auf die deutsche Sprachfassung zu verzichten und diese in der englischsprachigen Fassung zu veröffentlichen.

Es wird auf die Möglichkeit hingewiesen, dass einige Elemente dieses Dokuments Patentrechte berühren können. IEC ist nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

Documents
Document Sharing Hub

Full document details: E DIN EN IEC 62541-6:2019-09

Änderungen

Gegenüber DIN EN 62541-6:2015-11 wurden folgende Änderungen vorgenommen:

a) Kodierungen:
   - JSON-Kodierung für PubSub (nicht reversibel) hinzugefügt;
   - JSON-Kodierung für Client/Server (reversibel) hinzugefügt;
   - Unterstützung für optionale Felder in Strukturen hinzugefügt;
   - Unterstützung für Verbindungen hinzugefügt;

b) Transportabbildungen:
   - sichere WebSocket-Verbindung – WSS hinzugefügt;
   - Unterstützung für Rückwärtskonnektivität hinzugefügt;
   - Unterstützung für sitzungslosen Dienstaufruf in HTTPS hinzugefügt;

c) veralteter Transport (fehlende Unterstützung bei den meisten Plattformen):
   - SOAP/HTTP mit WS-SecureConversation (alle Kodierungen);

d) Abbildung für JSON-Web-Tokens hinzugefügt;

e) Unterstützung für Verbindungen zum Schema NodeSet hinzugefügt;

f) Batchoperationen zum Hinzufügen/Löschen von Knoten im Schema KnotenEinrichtung hinzugefügt;

g) Unterstützung für mehrdimensionale Felder außerhalb von Varianten hinzugefügt;

h) binäre Darstellung für Dezimaldatentypen hinzugefügt;

i) Abbildung für einen OAuth2-Autorisierungsrahmen hinzugefügt.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

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 legt die Abbildung der OPC Unified Architecture (OPC-UA) (vereinheitlichte Architektur der plattformunabhängigen Kommunikation) zwischen dem in IEC TR 62541-2 beschriebenen Sicherheitsmodell, den in IEC 62541-4 beschriebenen abstrakten Dienstdefinitionen, den in IEC 62541-5 beschriebenen Datenstrukturen und den physikalischen Netzwerkprotokollen fest, welche zur Implementierung der OPC-UA-Spezifikation verwendet werden können.

3 Begriffe und Konventionen

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC TR 62541-1, IEC TR 62541-1, IEC TR 62541-2, IEC 62541-3 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
ZertifikatsZusammenfassung
(en: CertificateDigest)
kurzer Bezeichner zur eindeutigen Bezeichnung eines X.509v3-*Zertifikats*

Anmerkung 1 zum Begriff: Dabei handelt es sich um den SHA1-Hash der DER-kodierten Form des Zertifikats.

3.1.2
DatenKodierung
(en: DataEncoding)
Möglichkeit zum Serialisieren von OPC-UA-*Nachrichten* und -Datenstrukturen

3.1.3
EntwicklungsPlattform
(en: DevelopmentPlatform)
Paket aus Tools und/oder Programmiersprachen zur Erstellung von Software

3.1.4
Abbildung
(en: Mapping)
legt fest, wie eine OPC-UA-Eigenschaft mit einer spezifischen Technologie implementiert wird

Anmerkung 1 zum Begriff: Zum Beispiel ist OPC-UA-Binärkodierung eine *Abbildung*, die festlegt, wie OPC-UA-Datenstrukturen als Bytesequenzen serialisiert werden.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

3.1.5
SicherheitsProtokoll
(en: SecurityProtocol)
gewährleistet die Integrität und die Geheimhaltung von OPC-UA-*Nachrichten*, die zwischen OPC-UA-Anwendungen ausgetauscht werden

3.1.6
StackProfil
(en: StackProfile)
Kombination von *Abbildungen* (en: *Mapping*) von *DatenKodierungen* (en: *DataEncordings*), *SicherheitsProtokollen* (en: *SecurityProtocol*) und *TransportProtokollen* (en: *TransportProtocol*)

Anmerkung 1 zum Begriff: OPC-UA-Anwendungen implementieren ein oder mehrere StackProfile und können nur mit OPC-UA-Anwendungen kommunizieren, die mindestens ein identisches StackProfil unterstützen.

3.1.7
TransportVerbindung
(en: TransportConnection)
zwischen OPC-UA-Anwendungen festgelegte Vollduplex-Kommunikationsverbindung

Anmerkung 1 zum Begriff: Ein TCP/IP-Sockel ist ein Beispiel für eine *TransportVerbindung*.

3.1.8
TransportProtokoll
(en: TransportProtocol)
repräsentiert eine Möglichkeit zum Austausch von serialisierten OPC-UA-*Nachrichten* zwischen OPC-UA-Anwendungen

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

– Leerseite –

Documents
Document Sharing Hub

Full document details: E DIN EN IEC 62541-6:2019-09

CONTENTS

FIGURES 4
TABLES 5
FOREWORD 8
1 Scope 11
2 Normative references 11
3 Terms, definitions and conventions 13
3.1 Terms and definitions 13
3.2 Abbreviations and symbols 14
4 Overview 14
5 Data encoding 15
5.1 General 15
5.1.1 Overview 15
5.1.2 Built-in Types 15
5.1.3 Guid 16
5.1.4 ByteString 17
5.1.5 ExtensionObject 17
5.1.6 Variant 17
5.1.7 Decimal 18
5.2 OPC UA Binary 18
5.2.1 General 18
5.2.2 Built-in Types 19
5.2.3 Decimal 27
5.2.4 Enumerations 27
5.2.5 Arrays 27
5.2.6 Structures 27
5.2.7 Structures with optional fields 29
5.2.8 Unions 31
5.2.9 Messages 32
5.3 OPC UA XML 33
5.3.1 Built-in Types 33
5.3.2 Decimal 39
5.3.3 Enumerations 39
5.3.4 Arrays 40
5.3.5 Structures 40
5.3.6 Structures with optional fields 40
5.3.7 Unions 41
5.3.8 Messages 41
5.4 OPC UA JSON 41
5.4.1 General 41
5.4.2 Built-in Types 42
5.4.3 Decimal 46
5.4.4 Enumerations 46
5.4.5 Arrays 46
5.4.6 Structures 47
5.4.7 Structures with optional fields 47
5.4.8 Unions 48

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

5.4.9 Messages 48
6 Message Security Protocols 48
6.1 Security handshake 48
6.2 Certificates 50
6.2.1 General 50
6.2.2 Application Instance Certificate 50
6.2.3 Certificate Chains 51
6.3 Time synchronization 52
6.4 UTC and International Atomic Time (TAI) 52
6.5 Issued User Identity Tokens 52
6.5.1 Kerberos 52
6.5.2 JSON Web Token (JWT) 52
6.5.3 OAuth2 53
6.6 WS Secure Conversation 55
6.7 OPC UA Secure Conversation 55
6.7.1 Overview 55
6.7.2 MessageChunk structure 55
6.7.3 MessageChunks and error handling 59
6.7.4 Establishing a Secure Channel 59
6.7.5 Deriving keys 61
6.7.6 Verifying Message Security 62
7 Transport Protocols 63
7.1 OPC UA Connection Protocol 63
7.1.1 Overview 63
7.1.2 Message structure 63
7.1.3 Establishing a connection 66
7.1.4 Closing a connection 67
7.1.5 Error handling 68
7.2 OPC UA TCP 69
7.3 SOAP/HTTP 69
7.4 OPC UA HTTPS 69
7.4.1 Overview 69
7.4.2 Session-less Services 70
7.4.3 XML Encoding 71
7.4.4 OPC UA Binary Encoding 72
7.4.5 JSON Encoding 72
7.5 WebSockets 72
7.5.1 Overview 72
7.5.2 Protocol Mapping 73
7.5.3 Security 74
7.6 Well known addresses 74
8 Normative Contracts 75
8.1 OPC Binary Schema 75
8.2 XML Schema and WSDL 75
Annex A (normative) Constants 76
A.1 Attribute Ids 76
A.2 Status Codes 76
A.3 Numeric Node Ids 76

AVA Documents
Document Sharing Hub

Annex B (normative) OPC UA Nodeset 78
Annex C (normative) Type declarations for the OPC UA native Mapping 79
Annex D (normative) WSDL for the XML Mapping 80
D.1 XML Schema 80
D.2 WDSL Port Types 80
D.3 WSDL Bindings 80
Annex E (normative) Security settings management 81
E.1 Overview 81
E.2 SecuredApplication 82
E.3 CertificateIdentifier 85
E.4 CertificateStoreIdentifier 86
E.5 CertificateList 87
E.6 CertificateValidationOptions 87
Annex F (normative) Information Model XML Schema 88
F.1 Overview 88
F.2 UANodeSet 88
F.3 UANode 89
F.4 Reference 90
F.5 RolePermission 91
F.6 UAType 91
F.7 UAInstance 91
F.8 UAVariable 92
F.9 UAMethod 92
F.10 TranslationType 93
F.11 UADataType 94
F.12 DataTypeDefinition 94
F.13 DataTypeField 94
F.14 Variant 95
F.15 Example (Informative) 95
F.16 UANodeSetChanges 98
F.17 NodesToAdd 99
F.18 ReferencesToChange 99
F.19 ReferenceToChange 99
F.20 NodesToDelete 100
F.21 NodeToDelete 100
F.22 UANodeSetChangesStatus 100
F.23 NodeSetStatusList 100
F.24 NodeSetStatus 101

FIGURES

Figure 1 – The OPC UA Stack Overview 15
Figure 2 – Encoding Integers in a binary stream 19
Figure 3 – Encoding Floating Points in a binary stream 19
Figure 4 – Encoding Strings in a binary stream 20
Figure 5 – Encoding Guids in a binary stream 21
Figure 6 – Encoding XmlElement in a binary stream 21

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

Figure 7 – A String NodeId 22
Figure 8 – A Two Byte NodeId 22
Figure 9 – A Four Byte NodeId 23
Figure 10 – Security handshake 49
Figure 11 – OPC UA Secure Conversation MessageChunk 56
Figure 12 – OPC UA Connection Protocol Message structure 63
Figure 13 – Client initiated OPC UA Connection Protocol connection 66
Figure 14 – Server initiated OPC UA Connection Protocol connection 67
Figure 15 – Closing a OPC UA Connection Protocol connection 67
Figure 16 – Scenarios for the HTTPS Transport 70
Figure 19 – Setting up Communication over a WebSocket 73

TABLES

Table 1 – Built-in Data Types 16
Table 2 – Guid structure 16
Table 3 – Layout of Decimal 18
Table 4 – Supported Floating Point Types 19
Table 5 – NodeId components 21
Table 6 – NodeId Data Encoding values 22
Table 7 – Standard NodeId Binary Data Encoding 22
Table 8 – Two Byte NodeId Binary Data Encoding 22
Table 9 – Four Byte NodeId Binary Data Encoding 23
Table 10 – ExpandedNodeId Binary Data Encoding 23
Table 11 – DiagnosticInfo Binary Data Encoding 24
Table 12 – QualifiedName Binary Data Encoding 24
Table 13 – LocalizedText Binary Data Encoding 25
Table 14 – Extension Object Binary Data Encoding 25
Table 15 – Variant Binary Data Encoding 26
Table 16 – Data Value Binary Data Encoding 27
Table 17 – Sample OPC UA Binary Encoded structure 28
Table 17 – Sample OPC UA Binary Encoded Structure with optional fields 30
Table 18 – Sample OPC UA Binary Encoded Structure 32
Table 18 – XML Data Type Mappings for Integers 33
Table 19 – XML Data Type Mappings for Floating Points 33
Table 20 – Components of NodeId 35
Table 21 – Components of ExpandedNodeId 36
Table 22 – Components of Enumeration 39
Table 23 – JSON Object Definition for a NodeId 43
Table 24 – JSON Object Definition for an ExpandedNodeId 43
Table 25 – JSON Object Definition for a StatusCode 44
Table 26 – JSON Object Definition for a DiagnosticInfo 44
Table 27 – JSON Object Definition for a QualifiedName 45

AVA Documents
Document Sharing Hub

Table 28 – JSON Object Definition for a LocalizedText 45
Table 29 – JSON Object Definition for a ExtensionObject 45
Table 30 – JSON Object Definition for a Variant 45
Table 31 – JSON Object Definition for a DataValue 46
Table 32 – JSON Object Definition for a Decimal 46
Table 33 – JSON Object Definition for a *Structures* with Optional Fields 47
Table 34 – JSON Object Definition for a Union 48
Table 35 – SecurityPolicy 49
Table 36 – Application Instance Certificate 51
Table 37 – Kerberos UserTokenPolicy 52
Table 38 – JWT UserTokenPolicy 53
Table 39 – JWT IssuerEndpointUrl Definition 53
Table 40 – Access Token Claims 54
Table 41 – OPC UA Secure Conversation Message header 56
Table 42 – Asymmetric algorithm Security header 57
Table 43 – Symmetric algorithm Security header 58
Table 44 – Sequence header 58
Table 45 – OPC UA Secure Conversation Message footer 58
Table 46 – OPC UA Secure Conversation Message abort body 59
Table 47 – OPC UA Secure Conversation OpenSecureChannel Service 60
Table 48 – PRF inputs for RSA based SecurityPolicies 61
Table 49 – Cryptography key generation parameters 61
Table 50 – OPC UA Connection Protocol Message header 64
Table 51 – OPC UA Connection Protocol Hello Message 64
Table 52 – OPC UA Connection Protocol Acknowledge Message 65
Table 53 – OPC UA Connection Protocol Error Message 65
Table 54 – OPC UA Connection Protocol ReverseHello Message 65
Table 55 – OPC UA Connection Protocol error codes 68
Table 57 – WebSocket Protocols Mappings 73
Table 58 – Well known addresses for Local Discovery Servers 74
Table A.1 – Identifiers assigned to Attributes 76
Table E.1 – SecuredApplication 82
Table E.2 – CertificateIdentifier 85
Table E.3 – Structured directory store 86
Table E.4 – CertificateStoreIdentifier 86
Table E.5 – CertificateList 87
Table E.6 – CertificateValidationOptions 87
Table F.1 – UANodeSet 89
Table F.2 – UANode 90
Table F.3 – Reference 91
Table F.4 – RolePermission 91
Table F.5 – UANodeSet Type Nodes 91
Table F.6 – UANodeSet Instance Nodes 91

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

Table F.7 – UAInstance 92
Table F.8 – UAVariable 92
Table F.9 – UAMethod 93
Table F.10 – TranslationType 94
Table F.11 – UADataType 94
Table F.12 – DataTypeDefinition 94
Table F.13 – DataTypeField 95
Table F.14 – UANodeSetChanges 98
Table F.15 – NodesToAdd 99
Table F.16 – ReferencesToChange 99
Table F.17 – ReferencesToChange 99
Table F.18 – NodesToDelete 100
Table F.19 – ReferencesToChange 100
Table F.20 – UANodeSetChangesStatus 100
Table F.21 – NodeSetStatusList 101
Table F.22 – NodeSetStatus 101

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

OPC UNIFIED ARCHITECTURE –

Part 6: Mappings

FOREWORD

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

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

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

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

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

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

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

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

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

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

IEC 62541-6 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this technical report is based on the following documents:

Enquiry draft	Report on voting
65E/XX/DTR	65E/XX/RVC

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

This third edition cancels and replaces the second edition of IEC 62541, published in 2015.

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

AVA Documents
Document Sharing Hub

a) Encodings:
- Added JSON encoding for PubSub (non-reversible)
- Added JSON encoding for Client/Server (reversible)
- Added support for optional fields in structures
- Added support for Unions

b) Transport mappings:
- Added WebSocket secure connection - WSS
- Added support for reverse connectivity.
- Added support for session-less service invocation in HTTPS

c) Deprecated Transport (missing support on most platforms):
- SOAP/HTTP with WS-SecureConversation (all encodings)

d) Added mapping for JSON Web Token
e) Added support for Unions to NodeSet Schema
f) Added batch operations to add/delete nodes to NodeSet Schema
g) Added support for multi-dimensional arrays outside of Variants
h) Added binary representation for Decimal data types
i) Added mapping for an OAuth2 Authorization Framework

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

Throughout this document and the referenced other Parts of the series, certain document conventions are used:

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

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

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

A list of all parts of the IEC 62541 series, published under the general title *OPC Unified Architecture*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

The National Committees are requested to note that for this publication the stability date is 2021.

AVA Documents Document Sharing Hub

Full document details: E DIN EN IEC 62541-6:2019-09

93 THIS TEXT IS INCLUDED FOR THE INFORMATION OF THE NATIONAL COMMITTEES AND WILL BE DELETED AT
94 THE PUBLICATION STAGE.

95 A bilingual version of this publication may be issued at a later date.

96

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

97

98

99

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

OPC Unified Architecture Specification

Part 6: Mappings

1 Scope

This part specifies the OPC Unified Architecture (OPC UA) mapping between the security model described in IEC TR 62541-2, the abstract service definitions, described in IEC 62541-4, the data structures defined in IEC 62541-5 and the physical network protocols that can be used to implement the OPC UA specification.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 TR 62541-2, *OPC Unified Architecture – Part 2: Security Model*

IEC 62541-3, *OPC Unified Architecture – Part 3: Address Space Model*

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

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

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

IEC 62541-12, *OPC Unified Architecture – Part 12: Discovery and Global Services*

IEC 62541-14, *OPC Unified Architecture – Part 14: PubSub*

XML Schema Part 1: XML Schema Part 1: Structures
http://www.w3.org/TR/xmlschema-1/
XML Schema Part 2: XML Schema Part 2: Datatypes
http://www.w3.org/TR/xmlschema-2/
SOAP Part 1: SOAP Version 1.2 Part 1: Messaging Framework
http://www.w3.org/TR/soap12-part1/
SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts
http://www.w3.org/TR/soap12-part2/
XML Encryption: XML Encryption Syntax and Processing
http://www.w3.org/TR/xmlenc-core/
XML Signature: XML-Signature Syntax and Processing
http://www.w3.org/TR/xmldsig-core/
WS Security: SOAP Message Security 1.1
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
WS Addressing: Web Services Addressing (WS-Addressing)
http://www.w3.org/Submission/ws-addressing/

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

SSL/TLS: RFC 5246 – The TLS Protocol Version 1.2
http://tools.ietf.org/html/rfc5246.txt
X.509 v3: X.509 Public Key Certificate Infrastructure
http://www.itu.int/rec/T-REC-X.509-200003-I/e
HTTP: RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1
http://www.ietf.org/rfc/rfc2616.txt
HTTPS: RFC 2818 – HTTP Over TLS
http://www.ietf.org/rfc/rfc2818.txt
Base64: RFC 3548 – The Base16, Base32, and Base64 Data Encodings
http://www.ietf.org/rfc/rfc3548.txt
X690: ITU-T X.690 – Basic (BER), Canonical (CER) and Distinguished (DER) Encoding Rules
http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf
X200: ITU-T X.200 – Open Systems Interconnection – Basic Reference Model
http://www.itu.int/rec/T-REC-X.200-199407-I/en
IEEE-754: Standard for Binary Floating-Point Arithmetic
http://grouper.ieee.org/groups/754/
HMAC: HMAC – Keyed-Hashing for Message Authentication
http://www.ietf.org/rfc/rfc2104.txt
PKCS #1: PKCS #1 – RSA Cryptography Specifications Version 2.0
http://www.ietf.org/rfc/rfc2437.txt
PKCS #12: PKCS 12: Personal Information Exchange Syntax
http://www.ietf.org/rfc/rfc7292.txt
FIPS 180-2: Secure Hash Standard (SHA)
http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
FIPS 197: Advanced Encryption Standard (AES)
http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf
UTF-8: UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc3629.txt
RFC 3280: RFC 3280 – X.509 Public Key Infrastructure Certificate and CRL Profile
http://www.ietf.org/rfc/rfc3280.txt
RFC 4514: RFC 4514 – LDAP: String Representation of Distinguished Names
http://www.ietf.org/rfc/rfc4514.txt
NTP: RFC 1305 – Network Time Protocol (Version 3)
http://www.ietf.org/rfc/rfc1305.txt
Kerberos: WS Security Kerberos Token Profile 1.1
http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf
RFC 1738: RFC 1738 - Uniform Resource Locators (URL)
http://www.ietf.org/rfc/rfc1738.txt
RFC 2141: RFC 2141 - URN Syntax
http://www.ietf.org/rfc/rfc2141.txt
RFC 6455: RFC 6455 - The WebSocket Protocol
http://www.ietf.org/rfc/rfc6455.txt
RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

http://www.ietf.org/rfc/rfc7159.txt

RFC 7523: JSON Web Token (JWT) Profile for OAuth 2.0

https://tools.ietf.org/rfc/rfc7523.txt

RFC 6749: The OAuth 2.0 Authorization Framework

http://www.ietf.org/rfc/rfc6749.txt

OpenID-Core: OpenID Connect Core 1.0

http://openid.net/specs/openid-connect-core-1_0.html

OpenID-Discovery: OpenID Connect Discovery 1.0

https://openid.net/specs/openid-connect-discovery-1_0.html

RFC 6960: RFC 6960 - Online Certificate Status Protocol - OCSP

https://tools.ietf.org/rfc/rfc6960.txt

3 Terms, definitions and conventions

3.1 Terms and definitions

For the purposes of this document the terms and definitions given in IEC TR 62541-1 IEC TR 62541-1, IEC TR 62541-2 and IEC 62541-3 as well as the following apply.

3.1.1
CertificateDigest
a short identifer used to uniquely identify an X.509v3 *Certificate*.

Note 1 to entry: This is the SHA1 hash of DER encoded form of the *Certificate*.

3.1.2
DataEncoding
a way to serialize OPC UA *Messages* and data structures.

3.1.3
DevelopmentPlatform
a suite a tools and/or programming languages used to create software.

3.1.4
Mapping
specifies how to implement an OPC UA feature with a specific technology.

Note 1 to entry: For example, the OPC UA Binary Encoding is a *Mapping* that specifies how to serialize OPC UA data structures as sequences of bytes.

3.1.5
SecurityProtocol
ensures the integrity and privacy of UA *Messages* that are exchanged between OPC UA applications

3.1.6
StackProfile
a combination of *DataEncodings, SecurityProtocol* and *TransportProtocol Mappings*

Note 1 to entry: OPC UA applications implement one or more *StackProfiles* and can only communicate with OPC UA applications that support a *StackProfile* that they support.

3.1.7
TransportConnection
a full-duplex communication link established between OPC UA applications.

Note 1 to entry: A TCP/IP socket is an example of a *TransportConnection*.

3.1.8
TransportProtocol
a way to exchange serialized OPC UA *Messages* between OPC UA applications

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

3.2 Abbreviations and symbols

API	Application Programming Interface
ASN.1	Abstract Syntax Notation #1 (used in X690)
CSV	Comma Separated Value (File Format)
ECC	Elliptic Curve Cryptography
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IPSec	Internet Protocol Security
OID	Object Identifier (used with ASN.1)
RSA	Rivest, Shamir and Adleman [Public Key Encryption System]
SHA1	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer (Defined in SSL/TLS)
TCP	Transmission Control Protocol
TLS	Transport Layer Security (Defined in SSL/TLS)
UA	Unified Architecture
UACP	OPC UA Connection Protocol
UASC	OPC UA Secure Conversation
WS-*	XML Web Services Specifications
XML	Extensible Markup Language

4 Overview

Other parts of this series of standards are written to be independent of the technology used for implementation. This approach means OPC UA is a flexible specification that will continue to be applicable as technology evolves. On the other hand, this approach means that it is not possible to build an OPC UA application with the information contained in IEC TR 62541-1 through to IEC 62541-5 because important implementation details have been left out.

This standard defines *Mappings* between the abstract specifications and technologies that can be used to implement them. The *Mappings* are organized into three groups: *DataEncodings, SecurityProtocols* and *TransportProtocols*. Different *Mappings* are combined together to create *StackProfiles*. All OPC UA applications shall implement at least one *StackProfile* and can only communicate with other OPC UA applications that implement the same *StackProfile.*

This standard defines the *DataEncodings* in Clause 5, the *SecurityProtocols* in Clause 5.4 and the *TransportProtocols* in 6.7.6. The *StackProfiles* are defined in IEC 62541-7.

All communication between OPC UA applications is based on the exchange of *Messages*. The parameters contained in the *Messages* are defined in IEC 62541-4; however, their format is specified by the *DataEncoding* and *TransportProtocol*. For this reason, each *Message* defined in IEC 62541-4 shall have a normative description which specifies exactly what shall be put on the wire. The normative descriptions are defined in the appendices.

A *Stack* is a collection of software libraries that implement one or more *StackProfiles*. The interface between an OPC UA application and the *Stack* is a non-normative API which hides the details of the *Stack* implementation. An API depends on a specific *DevelopmentPlatform*. Note that the datatypes exposed in the API for a *DevelopmentPlatform* may not match the datatypes defined by the specification because of limitations of the *DevelopmentPlatform.* For example, Java does not support an unsigned integer which means that any Java API will need to map unsigned integers onto a signed integer type.

Figure 1 illustrates the relationships between the different concepts defined in this standard.

AVA Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

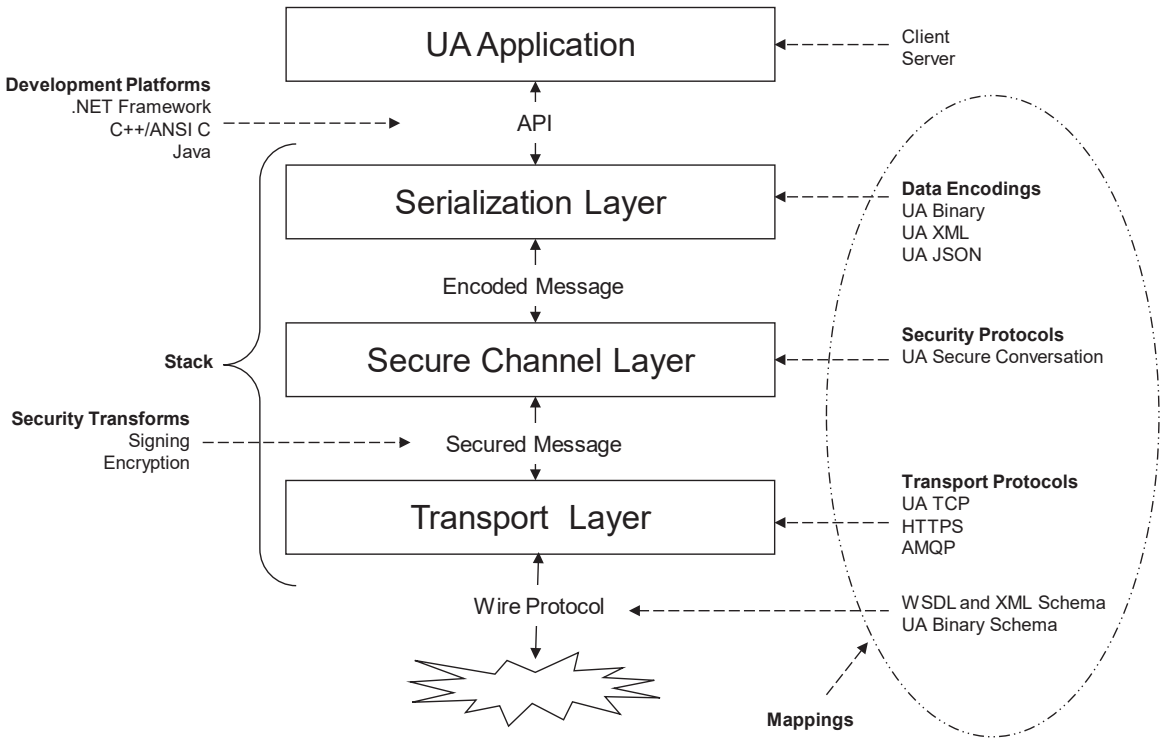

Figure 1 – The OPC UA Stack Overview

The layers described in this specification do not correspond to layers in the OSI 7-layer model [X]. Each OPC UA *StackProfile* should be treated as a single Layer 7 (application) protocol that is built on an existing Layer 5, 6 or 7 protocol such as TCP/IP, TLS or HTTP. The *SecureChannel* layer is always present even if the *SecurityMode* is *None*. In this situation, no security is applied but the *SecurityProtocol* implementation shall maintain a logical channel with a unique identifier. Users and administrators are expected to understand that a *SecureChannel* with *SecurityMode* set to *None* cannot be trusted unless the application is operating on a physically secure network or a low-level protocol such as IPSec is being used.

5 Data encoding

5.1 General

5.1.1 Overview

This standard defines three data encodings: OPC UA Binary, OPC UA XML and OPC UA JSON. It describes how to construct *Messages* using each of these encodings.

5.1.2 Built-in Types

All OPC UA *DataEncodings* are based on rules that are defined for a standard set of built-in types. These built-in types are then used to construct structures, arrays and *Messages*. The built-in types are described in Table 1.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

298 **Table 1 – Built-in Data Types**

ID	Name	Description
1	Boolean	A two-state logical value (true or false).
2	SByte	An integer value between −128 and 127 inclusive.
3	Byte	An integer value between 0 and 255 inclusive.
4	Int16	An integer value between −32 768 and 32 767 inclusive.
5	UInt16	An integer value between 0 and 65 535 inclusive.
6	Int32	An integer value between −2 147 483 648 and 2 147 483 647 inclusive.
7	UInt32	An integer value between 0 and 4 294 967 295 inclusive.
8	Int64	An integer value between −9 223 372 036 854 775 808 and 9 223 372 036 854 775 807 inclusive.
9	UInt64	An integer value between 0 and 18 446 744 073 709 551 615 inclusive.
10	Float	An IEEE single precision (32 bit) floating point value.
11	Double	An IEEE double precision (64 bit) floating point value.
12	String	A sequence of Unicode characters.
13	DateTime	An instance in time.
14	Guid	A 16-byte value that can be used as a globally unique identifier.
15	ByteString	A sequence of octets.
16	XmlElement	An XML element.
17	NodeId	An identifier for a node in the address space of an OPC UA *Server*.
18	ExpandedNodeId	A NodeId that allows the namespace URI to be specified instead of an index.
19	StatusCode	A numeric identifier for an error or condition that is associated with a value or an operation.
20	QualifiedName	A name qualified by a namespace.
21	LocalizedText	Human readable text with an optional locale identifier.
22	ExtensionObject	A structure that contains an application specific data type that may not be recognized by the receiver.
23	DataValue	A data value with an associated status code and timestamps.
24	Variant	A union of all of the types specified above.
25	DiagnosticInfo	A structure that contains detailed error and diagnostic information associated with a StatusCode.

299

300 Most of these data types are the same as the abstract types defined in IEC 62541-3 and
301 IEC 62541-4. However, the *ExtensionObject* and *Variant* types are defined in this standard. In
302 addition, this standard defines a representation for the *Guid* type defined in IEC 62541-3.

303 ### 5.1.3 Guid

304 A *Guid* is a 16-byte globally unique identifier with the layout shown in Table 2.

305 **Table 2 – Guid structure**

Component	Data Type
Data1	UInt32
Data2	UInt16
Data3	UInt16
Data4	Byte [8]

306

307 *Guid* values may be represented as a string in this form:

308 `<Data1>-<Data2>-<Data3>-<Data4[0:1]>-<Data4[2:7]>`

309 Where Data1 is 8 characters wide, Data2 and Data3 are 4 characters wide and each *Byte* in
310 Data4 is 2 characters wide. Each value is formatted as a hexadecimal number with padded
311 zeros. A typical *Guid* value would look like this when formatted as a string:

312 `C496578A-0DFE-4B8F-870A-745238C6AEAE`
313

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

5.1.4 ByteString

A *ByteString* is structurally the same as a one-dimensional array of *Byte*. It is represented as a distinct built-in data type because it allows encoders to optimize the transmission of the value. However, some *DevelopmentPlatforms* will not be able to preserve the distinction between a *ByteString* and a one-dimensional array of *Byte*.

If a decoder for *DevelopmentPlatform* cannot preserve the distinction it shall convert all one-dimensional arrays of *Byte* to *ByteStrings*.

Each element in a one-dimensional array of *ByteString* can have a different length which means is structurally different from a two-dimensional array of *Byte* where the length of each dimension is the same. This means decoders shall preserve the distinction between two or more dimension arrays of *Byte* and one or more dimension arrays of *ByteString*.

If a *DevelopmentPlatform* does not support unsigned integers, then it will have to represent *ByteStrings* as arrays of *SByte*. In this case, the requirements for *Byte* would then apply to *SByte*.

5.1.5 ExtensionObject

An *ExtensionObject* is a container for any *Structured DataTypes* which cannot be encoded as one of the other built-in data types. The *ExtensionObject* contains a complex value serialized as a sequence of bytes or as an XML element. It also contains an identifier which indicates what data it contains and how it is encoded.

Structured DataTypes are represented in a *Server* address space as sub-types of the *Structure DataType*. The *DataEncodings* available for any given *Structured DataTypes* are represented as a *DataTypeEncoding Object* in the *Server AddressSpace*. The *NodeId* for the *DataTypeEncoding Object* is the identifier stored in the *ExtensionObject*. IEC 62541-3 describes how *DataTypeEncoding Nodes* are related to other *Nodes* of the *AddressSpace*.

Server implementers should use namespace qualified numeric *NodeIds* for any *DataTypeEncoding Objects* they define. This will minimize the overhead introduced by packing *Structured DataType* values into an *ExtensionObject*.

ExtensionObjects and Variants allow unlimited nesting which could result in stack overflow errors even if the message size is less than the maximum allowed. Decoders shall support at least 100 nesting levels. Decoders shall report an error if the number of nesting levels exceeds what it supports.

5.1.6 Variant

A *Variant* is a union of all built-in data types including an *ExtensionObject*. *Variants* can also contain arrays of any of these built-in types. *Variants* are used to store any value or parameter with a data type of *BaseDataType* or one of its subtypes.

Variants can be empty. An empty *Variant* is described as having a null value and should be treated like a null column in a SQL database. A null value in a *Variant* may not be the same as a null value for data types that support nulls such as *Strings*. Some *DevelopmentPlatforms* may not be able to preserve the distinction between a null for a *DataType* and a null for a *Variant*, therefore, applications shall not rely on this distinction. This requirement also means that if an *Attribute* supports the writing of a null value it shall also support writing of an empty *Variant* and vice versa.

Variants can contain arrays of *Variants* but they cannot directly contain another *Variant*.

DiagnosticInfo types only have meaning when returned in a response message with an associated *StatusCode* and table of strings. As a result, *Variants* cannot contain instances of *DiagnosticInfo.*

Values of *Attributes* are always returned in instances of *DataValues*. Therefore, the *DataType* of an *Attribute* cannot be a *DataValue*. *Variants* can contain *DataValue* when used in other contexts such as *Method Arguments* or *PubSub Messages*. The *Variant* in a *DataValue* cannot, directly or indirectly, contain another *DataValue*.

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

Variables with a *DataType* of *BaseDataType* are mapped to a *Variant*, however, the *ValueRank* and *ArrayDimensions Attributes* place restrictions on what is allowed in the *Variant*. For example, if the *ValueRank* is *Scalar* then the *Variant* may only contain scalar values.

ExtensionObjects and Variants allow unlimited nesting which could result in stack overflow errors even if the message size is less than the maximum allowed. Decoders shall support at least 100 nesting levels. Decoders shall report an error if the number of nesting levels exceeds what it supports.

5.1.7 Decimal

A *Decimal* is a high-precision signed decimal number. It consists of an arbitrary precision integer unscaled value and an integer scale. The scale is the power of ten that is applied to the unscaled value.

A *Decimal* has the fields described in Table 3.

Table 3 – Layout of Decimal

Field	Type	Description
TypeId	NodeId	The identifier for the *Decimal DataType*.
Encoding	Byte	This value is always 1.
Length	Int32	The length of the *Decimal*. If the length is less than or equal to 0 then the *Decimal* value is 0.
Scale	Int16	A signed integer representing the power of ten used to scale the value. i.e. the decimal number of the value * 10^{-scale} The integer is encoded starting with the least significant bit.
Value	Byte [*]	A 2-complement signed integer representing the unscaled value. The number of bits is inferred from the length of the *length* field. If the number of bits is 0 then the value is 0. The integer is encoded with the least significant byte first.

When a *Decimal* is encoded in a *Variant* the built-in type is set to *ExtensionObject*. Decoders that do not understand the *Decimal* type shall treat it like any other unknown *Structure* and pass it on to the application. Decoders that do understand the *Decimal* can parse the value and use any construct that is suitable for the *DevelopmentPlatform*.

If a *Decimal* is embedded in another *Structure* then the *DataTypeDefinition* for the field shall specify the *NodeId* of the *Decimal Node* as the *DataType*. If a *Server* publishes an OPC Binary type description for the *Structure* then the type description shall set the *DataType* for the field to *ExtensionObject*.

5.2 OPC UA Binary

5.2.1 General

The OPC UA *Binary DataEncoding* is a data format developed to meet the performance needs of OPC UA applications. This format is designed primarily for fast encoding and decoding, however, the size of the encoded data on the wire was also a consideration.

The OPC UA *Binary DataEncoding* relies on several primitive data types with clearly defined encoding rules that can be sequentially written to or read from a binary stream. A structure is encoded by sequentially writing the encoded form of each field. If a given field is also a structure, then the values of its fields are written sequentially before writing the next field in the containing structure. All fields shall be written to the stream even if they contain null values. The encodings for each primitive type specify how to encode either a null or a default value for the type.

The OPC UA *Binary DataEncoding* does not include any type or field name information because all OPC UA applications are expected to have advance knowledge of the services and structures that they support. An exception is an *ExtensionObject* which provides an identifier and a size for the *Structured DataType* structure it represents. This allows a decoder to skip over types that it does not recognize.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

5.2.2 Built-in Types

5.2.2.1 Boolean

A *Boolean* value shall be encoded as a single byte where a value of 0 (zero) is false and any non-zero value is true.

Encoders shall use the value of 1 to indicate a true value; however, decoders shall treat any non-zero value as true.

5.2.2.2 Integer

All integer types shall be encoded as little endian values where the least significant byte appears first in the stream.

Figure 2 illustrates how value 1 000 000 000 (Hex: 3B9ACA00) should be encoded as a 32-bit integer in the stream.

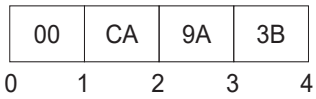

Figure 2 – Encoding Integers in a binary stream

5.2.2.3 Floating Point

All floating-point values shall be encoded with the appropriate IEEE-754 binary representation which has three basic components: the sign, the exponent, and the fraction. The bit ranges assigned to each component depend on the width of the type. Table 4 lists the bit ranges for the supported floating point types.

Table 4 – Supported Floating Point Types

Name	Width (bits)	Fraction	Exponent	Sign
Float	32	0-22	23-30	31
Double	64	0-51	52-62	63

In addition, the order of bytes in the stream is significant. All floating point values shall be encoded with the least significant byte appearing first (i.e. little endian).

Figure 3 illustrates how the value −6,5 (Hex: C0D00000) should be encoded as a *Float*.

The floating-point type supports positive and negative infinity and not-a-number (NaN). The IEEE specification allows for multiple NaN variants; however, the encoders/decoders may not preserve the distinction. Encoders shall encode a NaN value as an IEEE quiet-NAN (000000000000F8FF) or (0000C0FF). Any unsupported types such as denormalized numbers shall also be encoded as an IEEE quiet-NAN. Any test for equality between NaN values always fails.

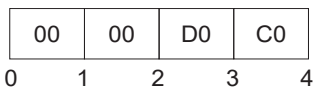

Figure 3 – Encoding Floating Points in a binary stream

5.2.2.4 String

All *String* values are encoded as a sequence of UTF-8 characters without a null terminator and preceded by the length in bytes.

The length in bytes is encoded as *Int32*. A value of −1 is used to indicate a 'null' string.

AVA Documents Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

Figure 4 illustrates how the multilingual string "水Boy" should be encoded in a byte stream.

Length				水			B	o	y
06	00	00	00	E6	B0	B4	42	6F	79
0	1	2	3	4	5	6	7	8	9	10

Figure 4 – Encoding Strings in a binary stream

5.2.2.5 DateTime

A *DateTime* value shall be encoded as a 64-bit signed integer (see Clause 5.2.2.2) which represents the number of 100 nanosecond intervals since January 1, 1601 (UTC).

Not all *DevelopmentPlatforms* will be able to represent the full range of dates and times that can be represented with this *DataEncoding*. For example, the UNIX time_t structure only has a 1 second resolution and cannot represent dates prior to 1970. For this reason, a number of rules shall be applied when dealing with date/time values that exceed the dynamic range of a *DevelopmentPlatform*. These rules are:

j) A date/time value is encoded as 0 if either
   1) The value is equal to or earlier than 1601-01-01 12:00AM UTC.
   2) The value is the earliest date that can be represented with the *DevelopmentPlatform*'s encoding.

k) A date/time is encoded as the maximum value for an *Int64* if either
   3) The value is equal to or greater than 9999-12-31 11:59:59PM UTC,
   4) The value is the latest date that can be represented with the *DevelopmentPlatform*'s encoding.

l) A date/time is decoded as the earliest time that can be represented on the platform if either
   5) The encoded value is 0,
   6) The encoded value represents a time earlier than the earliest time that can be represented with the *DevelopmentPlatform*'s encoding.

m) A date/time is decoded as the latest time that can be represented on the platform if either
   7) The encoded value is the maximum value for an *Int64*,
   8) The encoded value represents a time later than the latest time that can be represented with the *DevelopmentPlatform*'s encoding.

These rules imply that the earliest and latest times that can be represented on a given platform are invalid date/time values and should be treated that way by applications.

A decoder shall truncate the value if a decoder encounters a *DateTime* value with a resolution that is greater than the resolution supported on the *DevelopmentPlatform*.

5.2.2.6 Guid

A *Guid* is encoded in a structure as shown in Table 2. Fields are encoded sequentially according to the data type for field.

Figure 5 illustrates how the *Guid* "72962B91-FA75-4AE6-8D28-B404DC7DAF63" should be encoded in a byte stream.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

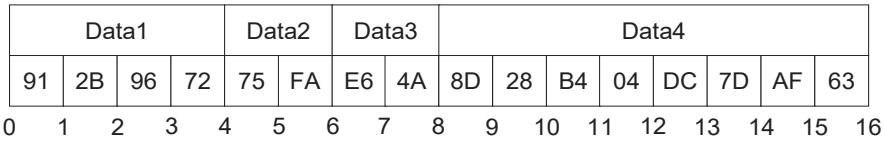

Figure 5 – Encoding Guids in a binary stream

5.2.2.7 ByteString

A *ByteString* is encoded as sequence of bytes preceded by its length in bytes. The length is encoded as a 32-bit signed integer as described above.

If the length of the byte string is −1 then the byte string is 'null'.

5.2.2.8 XmlElement

An *XmlElement* is an XML fragment serialized as UTF-8 string and then encoded as *ByteString.*

Figure 6 illustrates how the *XmlElement* "<A>Hot水</A>" should be encoded in a byte stream.

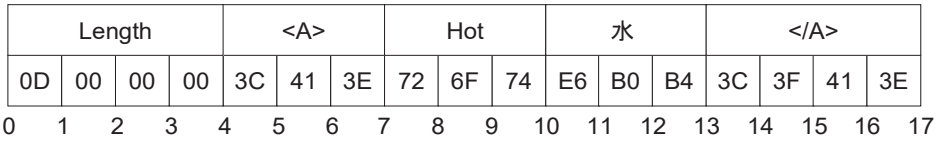

Figure 6 – Encoding XmlElement in a binary stream

A decoder may choose to parse the XML after decoding; if an unrecoverable parsing error occurs then the decoder should try to continue processing the stream. For example, if the *XmlElement* is the body of a *Variant* or an element in an array which is the body of a *Variant* then this error can be reported by setting value of the *Variant* to the *StatusCode Bad_DecodingError*.

5.2.2.9 NodeId

The components of a *NodeId* are described the Table 5.

Table 5 – NodeId components

Name	Data Type	Description
Namespace	UInt16	The index for a namespace URI. An index of 0 is used for OPC UA defined *NodeIds*.
IdentifierType	Enumeration	The format and data type of the identifier. The value may be one of the following: NUMERIC - the value is an *UInteger;* STRING - the value is *String*; GUID - the value is a *Guid*; OPAQUE - the value is a *ByteString*;
Value	*	The identifier for a node in the address space of an OPC UA *Server*.

The *DataEncoding* of a *NodeId* varies according to the contents of the instance. For that reason, the first byte of the encoded form indicates the format of the rest of the encoded *NodeId*. The possible *DataEncoding* formats are shown in Table 6. The tables that follow describe the structure of each possible format (they exclude the byte which indicates the format).

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

Table 6 – NodeId DataEncoding values

Name	Value	Description
Two Byte	0x00	A numeric value that fits into the two-byte representation.
Four Byte	0x01	A numeric value that fits into the four-byte representation.
Numeric	0x02	A numeric value that does not fit into the two or four byte representations.
String	0x03	A String value.
Guid	0x04	A Guid value.
ByteString	0x05	An opaque (ByteString) value.
NamespaceUri Flag	0x80	See discussion of *ExpandedNodeId* in 5.2.2.10.
ServerIndex Flag	0x40	See discussion of *ExpandedNodeId* in 5.2.2.10.

The standard *NodeId DataEncoding* has the structure shown in Table 7. The standard *DataEncoding* is used for all formats that do not have an explicit format defined.

Table 7 – Standard NodeId Binary DataEncoding

Name	Data Type	Description
Namespace	UInt16	The *NamespaceIndex*.
Identifier	*	The identifier which is encoded according to the following rules: NUMERIC UInt32 STRING String GUID Guid OPAQUE ByteString

An example of a String *NodeId* with Namespace = 1 and Identifier = “Hot水” is shown in Figure 7.

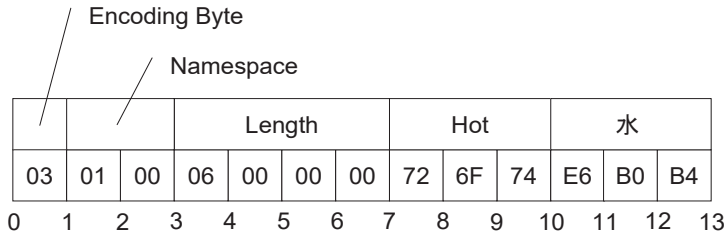

Figure 7 – A String NodeId

The Two Byte *NodeId DataEncoding* has the structure shown in Table 8.

Table 8 – Two Byte NodeId Binary DataEncoding

Name	Data Type	Description
Identifier	Byte	The *Namespace* is the default OPC UA namespace (i.e. 0). The *Identifier* Type is ‘Numeric’. The *Identifier* shall be in the range 0 to 255.

An example of a Two Byte *NodeId* with Identifier = 72 is shown in Figure 8.

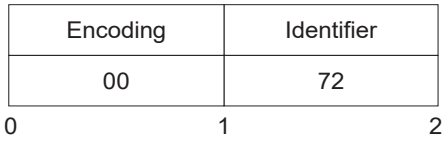

Figure 8 – A Two Byte NodeId

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

The Four Byte *NodeId DataEncoding* has the structure shown in Table 9.

Table 9 – Four Byte NodeId Binary DataEncoding

Name	Data Type	Description
Namespace	Byte	The *Namespace* shall be in the range 0 to 255.
Identifier	UInt16	The *Identifier* Type is 'Numeric'. The *Identifier* shall be an integer in the range 0 to 65 535.

An example of a Four Byte *NodeId* with Namespace = 5 and Identifier = 1 025 is shown in Figure 9.

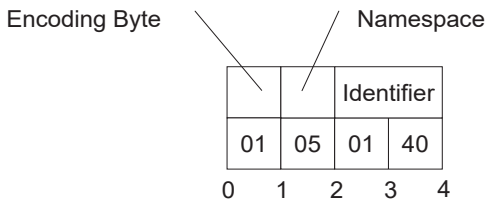

Figure 9 – A Four Byte NodeId

5.2.2.10 ExpandedNodeId

An *ExpandedNodeId* extends the *NodeId* structure by allowing the *NamespaceUri* to be explicitly specified instead of using the *NamespaceIndex*. The *NamespaceUri* is optional. If it is specified, then the *NamespaceIndex* inside the *NodeId* shall be ignored.

The *ExpandedNodeId* is encoded by first encoding a *NodeId* as described in 5.2.2.9 and then encoding *NamespaceUri* as a *String*.

An instance of an *ExpandedNodeId* may still use the *NamespaceIndex* instead of the *NamespaceUri*. In this case, the *NamespaceUri* is not encoded in the stream. The presence of the *NamespaceUri* in the stream is indicated by setting the *NamespaceUri* flag in the encoding format byte for the *NodeId*.

If the *NamespaceUri* is present, then the encoder shall encode the *NamespaceIndex* as 0 in the stream when the *NodeId* portion is encoded. The unused *NamespaceIndex* is included in the stream for consistency.

An *ExpandedNodeId* may also have a *ServerIndex* which is encoded as a *UInt32* after the *NamespaceUri.* The *ServerIndex* flag in the *NodeId* encoding byte indicates whether the *ServerIndex* is present in the stream. The *ServerIndex* is omitted if it is equal to zero.

The *ExpandedNodeId* encoding has the structure shown in Table 10.

Table 10 – ExpandedNodeId Binary DataEncoding

Name	Data Type	Description
NodeId	NodeId	The NamespaceUri and ServerIndex flags in the NodeId encoding indicate whether those fields are present in the stream.
NamespaceUri	String	Not present if null or Empty.
ServerIndex	UInt32	Not present if 0.

5.2.2.11 StatusCode

A *StatusCode* is encoded as a *UInt32*.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

548 #### 5.2.2.12 DiagnosticInfo

549 A *DiagnosticInfo* structure is described in IEC 62541-4. It specifies a number of fields that
550 could be missing. For that reason, the encoding uses a bit mask to indicate which fields are
551 actually present in the encoded form.

552 As described in IEC 62541-4, the *SymbolicId*, *NamespaceUri*, *LocalizedText* and *Locale* fields
553 are indexes in a string table which is returned in the response header. Only the index of the
554 corresponding string in the string table is encoded. An index of −1 indicates that there is no
555 value for the string.

556 *DiagnosticInfo* allows unlimited nesting which could result in stack overflow errors even if the
557 message size is less than the maximum allowed. Decoders shall support at least 100 nesting
558 levels. Decoders shall report an error if the number of nesting levels exceeds what it supports.

559 **Table 11 – DiagnosticInfo Binary DataEncoding**

Name	Data Type	Description
Encoding Mask	Byte	A bit mask that indicates which fields are present in the stream. The mask has the following bits: 0x01 Symbolic Id 0x02 Namespace 0x04 LocalizedText 0x08 Locale 0x10 Additional Info 0x20 InnerStatusCode 0x40 InnerDiagnosticInfo
SymbolicId	Int32	A symbolic name for the status code.
NamespaceUri	Int32	A namespace that qualifies the symbolic id.
Locale	Int32	The locale used for the localized text.
LocalizedText	Int32	A human readable summary of the status code.
Additional Info	String	Detailed application specific diagnostic information.
Inner StatusCode	StatusCode	A status code provided by an underlying system.
Inner DiagnosticInfo	DiagnosticInfo	Diagnostic info associated with the inner status code.

560

561 #### 5.2.2.13 QualifiedName

562 A *QualifiedName* structure is encoded as shown in Table 12.

563 The abstract *QualifiedName* structure is defined in IEC 62541-3.

564 **Table 12 – QualifiedName Binary DataEncoding**

Name	Data Type	Description
NamespaceIndex	UInt16	The namespace index.
Name	String	The name.

565

566 #### 5.2.2.14 LocalizedText

567 A *LocalizedText* structure contains two fields that could be missing. For that reason, the
568 encoding uses a bit mask to indicate which fields are actually present in the encoded form.

569 The abstract *LocalizedText* structure is defined in IEC 62541-3.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

570 **Table 13 – LocalizedText Binary DataEncoding**

Name	Data Type	Description
EncodingMask	Byte	A bit mask that indicates which fields are present in the stream. The mask has the following bits: 0x01 Locale 0x02 Text
Locale	String	The locale. Omitted is null or empty.
Text	String	The text in the specified locale. Omitted is null or empty.

571

572 ### 5.2.2.15 ExtensionObject

573 An *ExtensionObject* is encoded as sequence of bytes prefixed by the *NodeId* of its
574 *DataTypeEncoding* and the number of bytes encoded.

575 An *ExtensionObject* may be encoded by the application which means it is passed as a
576 *ByteString* or an *XmlElement* to the encoder. In this case, the encoder will be able to write the
577 number of bytes in the object before it encodes the bytes. However, an *ExtensionObject* may
578 know how to encode/decode itself which means the encoder shall calculate the number of
579 bytes before it encodes the object or it shall be able to seek backwards in the stream and
580 update the length after encoding the body.

581 When a decoder encounters an *ExtensionObject* it shall check if it recognizes the
582 *DataTypeEncoding* identifier. If it does, then it can call the appropriate function to decode the
583 object body. If the decoder does not recognize the type it shall use the *Encoding* to determine
584 if the body is a *ByteString* or an *XmlElement* and then decode the object body or treat it as
585 opaque data and skip over it.

586 The serialized form of an *ExtensionObject* is shown in Table 14.

587 **Table 14 – Extension Object Binary DataEncoding**

Name	Data Type	Description
TypeId	NodeId	The identifier for the *DataTypeEncoding* node in the *Server's AddressSpace*. *ExtensionObjects* defined by the OPC UA specification have a numeric node identifier assigned to them with a *NamespaceIndex* of 0. The numeric identifiers are defined in A.3. Decoders use this field to determine the syntax of the *Body*. For example, if this field is the *NodeId* of the *JSON Encoding Object* for a *DataType* then the *Body* is a *ByteString* containing a JSON document encoded as a UTF-8 string.
Encoding	Byte	An enumeration that indicates how the body is encoded. The parameter may have the following values: 0x00 No body is encoded. 0x01 The body is encoded as a ByteString. 0x02 The body is encoded as a XmlElement.
Length	Int32	The length of the object body. The length shall be specified if the body is encoded.
Body	Byte [*]	The object body. This field contains the raw bytes for ByteString bodies. For XmlElement bodies this field contains the XML encoded as a UTF-8 string without any null terminator. Some binary encoded structures may have a serialized length that is not a multiple of 8 bits. Encoders shall append 0 bits to ensure the serialized length is a multiple of 8 bits. Decoders that understand the serialized format shall ignore the padding bits.

588

589 *ExtensionObjects* are used in two contexts: as values contained in *Variant* structures or as
590 parameters in OPC UA *Messages*.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

A decoder may choose to parse an *XmlElement* body after decoding; if an unrecoverable parsing error occurs then the decoder should try to continue processing the stream. For example, if the *ExtensionObject* is the body of a *Variant* or an element in an array that is the body of *Variant* then this error can be reported by setting value of the *Variant* to the *StatusCode Bad_DecodingError*.

5.2.2.16 Variant

A *Variant* is a union of the built-in types.

The structure of a *Variant* is shown in Table 15.

Table 15 – Variant Binary DataEncoding

Name	Data Type	Description
EncodingMask	Byte	The type of data encoded in the stream. A value of 0 specifies a NULL and that no other fields are encoded. The mask has the following bits assigned: 0:5 Built-in Type Id (see Table 1). 6 True if the Array Dimensions field is encoded. 7 True if an array of values is encoded. The Built-in Type Ids 26 through 31 are not currently assigned but may be used in the future. Decoders shall accept these IDs, assume the *Value* contains a *ByteString* and pass both onto the application. Encoders shall not use these IDs.
ArrayLength	Int32	The number of elements in the array. This field is only present if the array bit is set in the encoding mask. Multi-dimensional arrays are encoded as a one-dimensional array and this field specifies the total number of elements. The original array can be reconstructed from the dimensions that are encoded after the value field. Higher rank dimensions are serialized first. For example, an array with dimensions [2,2,2] is written in this order: [0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]
Value	*	The value encoded according to its built-in data type. If the array bit is set in the encoding mask, then each element in the array is encoded sequentially. Since many types have variable length encoding each element shall be decoded in order. The value shall not be a *Variant* but it could be an array of *Variants*. Many implementation platforms do not distinguish between one dimensional Arrays of *Bytes* and *ByteStrings*. For this reason, decoders are allowed to automatically convert an Array of *Bytes* to a *ByteString*.
ArrayDimensions Length	Int32	The number of dimensions. This field is only present if the ArrayDimensions flag is set in the encoding mask.
ArrayDimensions	Int32[*]	The length of each dimension encoded as a sequence of Int32 values This field is only present if the ArrayDimensions flag is set in the encoding mask. The lower rank dimensions appear first in the array. All dimensions shall be specified and shall be greater than zero. If *ArrayDimensions* are inconsistent with the *ArrayLength* then the decoder shall stop and raise a *Bad_DecodingError*.

The types and their identifiers that can be encoded in a *Variant* are shown in Table 1.

5.2.2.17 DataValue

A *DataValue* is always preceded by a mask that indicates which fields are present in the stream.

The fields of a *DataValue* are described in Table 16.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09

606 **Table 16 – Data Value Binary DataEncoding**

Name	Data Type	Description
Encoding Mask	Byte	A bit mask that indicates which fields are present in the stream. The mask has the following bits: 0x01 False if the Value is *Null.* 0x02 False if the StatusCode is Good. 0x04 False if the Source Timestamp is *DateTime.MinValue*. 0x08 False if the *Server* Timestamp is *DateTime.MinValue*. 0x10 False if the Source Picoseconds is 0. 0x20 False if the *Server* Picoseconds is 0.
Value	Variant	The value. Not present if the Value bit in the EncodingMask is False.
Status	StatusCode	The status associated with the value. Not present if the StatusCode bit in the EncodingMask is False.
SourceTimestamp	DateTime	The source timestamp associated with the value. Not present if the SourceTimestamp bit in the EncodingMask is False.
SourcePicoSeconds	UInt16	The number of 10 picosecond intervals for the SourceTimestamp. Not present if the SourcePicoSeconds bit in the EncodingMask is False. If the source timestamp is missing the picoseconds are ignored.
ServerTimestamp	DateTime	The *Server* timestamp associated with the value. Not present if the ServerTimestamp bit in the EncodingMask is False.
ServerPicoSeconds	UInt16	The number of 10 picosecond intervals for the ServerTimestamp. Not present if the ServerPicoSeconds bit in the EncodingMask is False. If the *Server* timestamp is missing the picoseconds are ignored.

607

608 The *Picoseconds* fields store the difference between a high-resolution timestamp with a
609 resolution of 10 picoseconds and the *Timestamp* field value which only has a 100 ns
610 resolution. The *Picoseconds* fields shall contain values less than 10 000. The decoder shall
611 treat values greater than or equal to 10 000 as the value '9999'.

612 ### 5.2.3 Decimal

613 *Decimals* are encoded as described in 5.1.7.

614 A *Decimal* does not have a NULL value.

615 ### 5.2.4 Enumerations

616 Enumerations are encoded as *Int32* values.

617 An *Enumeration* does not have a NULL value.

618 ### 5.2.5 Arrays

619 One dimensional *Arrays* are encoded as a sequence of elements preceded by the number of
620 elements encoded as an *Int32* value. If an *Array* is null, then its length is encoded as −1. An
621 *Array* of zero length is different from an *Array* that is null so encoders and decoders shall
622 preserve this distinction.

623 Multi-dimensional *Arrays* are encoded as an *Int32 Array* containing the dimensions followed
624 by a list of all the values in the *Array*. The total number of values is equal to the product of the
625 dimensions. The number of values is 0 if one or more dimension is less than or equal to 0.
626 The process for reconstructing the multi-dimensional array is described in 5.2.2.16.

627 ### 5.2.6 Structures

628 *Structures* are encoded as a sequence of fields in the order that they appear in the definition.
629 The encoding for each field is determined by the built-in type for the field.

630 All fields specified in the structure shall be encoded. If optional fields exist in the structure
631 then see 5.2.7.

Documents
Document Sharing Hub
Full document details: E DIN EN IEC 62541-6:2019-09