

DIN EN 62541-6

ICS 35.240.50; 35.200

Einsprüche bis 2014-12-24
Vorgesehen als Ersatz für
DIN EN 62541-6:2012-06**Entwurf****OPC Unified Architecture –
Teil 6: Protokollabbildungen
(IEC 65E/377/CDV:2014);
Englische Fassung FprEN 62541-6:2014**OPC Unified Architecture –
Part 6: Mappings
(IEC 65E/377/CDV:2014);
English version FprEN 62541-6:2014Architecture Unifiée OPC –
Partie 6: Correspondances
(CEI 65E/377/CDV:2014);
Version anglaise FprEN 62541-6:2014**Anwendungswarnvermerk**

Dieser Norm-Entwurf mit Erscheinungsdatum 2014-10-24 wird der Öffentlichkeit zur Prüfung und
Stellungnahme vorgelegt.

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

Stellungnahmen werden erbeten

- vorzugsweise online im Norm-Entwurfs-Portal des DIN unter www.entwuerfe.din.de 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/stellungnahme 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 im 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 91 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Nationales Vorwort

Die Englische Fassung des europäischen Dokuments FprEN 62541-6:2014 „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/377/CDV:2014 „OPC Unified Architecture – Part 6: Mappings“ unverändert in den Entwurf FprEN 62541-6:2014 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 DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

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

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

Der Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen 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 Normentwurf erfüllt.

Da sich die Benutzer des vorliegenden Normentwurfs der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN 62541-6 veröffentlicht. Zu deren Abschnitt 3, Begriffe, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Normentwurfs hinzugefügt. Dieser Normentwurf 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. ISO [und/oder] IEC sind nicht dafür verantwortlich, einige oder alle diesbezüglichen Patentrechte zu identifizieren.

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 62541-6:2012-06 wurden folgende Änderungen vorgenommen:

- a) Einige Anwendungen müssen in einer Umgebung arbeiten, aus der heraus es keinen Zugang zu kryptographischen Bibliotheken gibt. Um dieses zu unterstützen, neuer HTTPS-Transport in 7.3 hinzugefügt;
- b) das padding byte ist nicht lang genug, um asymmetrische Schlüsselgrößen größer als 2048 bits zu handhaben. Ein zusätzliches padding byte in 6.7.2 hinzugefügt, um diesen Fall handhaben zu können;
- c) Fehler in den URIs der SOAP Optionen beseitigt;
- d) eine genormte Vorgehensweise zur Serialisierung von Knoten in einem Adressraum wurde benötigt. Daher das UANodeSet-Schema nach Anhang F hinzugefügt.

Nationaler Anhang NA (informativ)

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC/TR 62541-1, IEC/TR 62541-2 und IEC 62541-3 sowie die folgenden Begriffe.

3.1.1

Datenkodierung

(en: **Data Encoding**)

Datenkodierung ist ein Weg, OPC UA-Botschaften und Datenstrukturen zu serialisieren

3.1.2

Abbildungsvorschrift

(en: **Mapping**)

legt fest, wie eine OPC UA-Eigenschaft mit einer spezifischen Technologie implementiert wird

ANMERKUNG Zum Beispiel ist OPC UA-Binary Encoding eine Abbildungsvorschrift, die festlegt, wie OPC UA-Datenstrukturen als Bytesequenzen serialisiert werden.

3.1.3

Sicherheitsprotokoll

(en: **Security Protocol**)

gewährleistet die Integrität und die Geheimhaltung von OPC UA-Nachrichten, die zwischen OPC UA-Anwendungen ausgetauscht werden

3.1.4

Stackprofil

(en: **Stack Profile**)

Kombination von Datenkodierungen, Sicherheitsprotokoll und Transportprotokoll

ANMERKUNG OPC UA-Anwendungen implementieren ein oder mehrere Stackprofile und können mit OPC UA-Anwendungen kommunizieren, die mindestens ein identisches Stackprofil unterstützen.

3.1.5

Transportprotokoll

(en: **Transport Protocol**)

repräsentiert einen Weg zum Austausch von serialisierten OPC UA-Nachrichten zwischen OPC UA-Anwendungen

– Leerseite –

CONTENTS

1			
2			
3	FOREWORD	viii	
4	1 Scope	1	
5	2 Normative references	1	
6	3 Terms, definitions and conventions	3	
7	3.1 Terms and definitions	3	
8	3.2 Abbreviations and symbols	4	
9	4 Overview	4	
10	5 Data Encoding	5	
11	5.1 General	5	
12	5.1.1 Overview	5	
13	5.1.2 Built-in Types	5	
14	5.1.3 Guid	6	
15	5.1.4 ByteString	7	
16	5.1.5 ExtensionObject`	7	
17	5.1.6 Variant	7	
18	5.2 OPC UA Binary	8	
19	5.2.1 General	8	
20	5.2.2 Built-in Types	8	
21	5.2.3 Enumerations	17	
22	5.2.4 Arrays	17	
23	5.2.5 Structures	17	
24	5.2.6 Messages	18	
25	5.3 XML	18	
26	5.3.1 Built-in Types	18	
27	5.3.2 Enumerations	25	
28	5.3.3 Arrays	25	
29	5.3.4 Structures	26	
30	5.3.5 Messages	26	
31	6 Message Security Protocols	26	
32	6.1 Security Handshake	26	
33	6.2 Certificates	28	
34	6.2.1 General	28	
35	6.2.2 Application Instance Certificate	29	
36	6.2.3 Signed Software Certificate	30	
37	6.3 Time Synchronization	30	
38	6.4 UTC and International Atomic Time (TAI)	31	
39	6.5 Issued User Identity Tokens	31	
40	6.5.1 Kerberos	31	
41	6.6 WS Secure Conversation	31	
42	6.6.1 Overview	31	
43	6.6.2 Notation	33	
44	6.6.3 Request Security Token (RST/SCT)	33	
45	6.6.4 Request Security Token Response (RSTR/SCT)	34	
46	6.6.5 Using the SCT	35	

47	6.6.6	Cancelling Security Contexts	35
48	6.7	OPC UA Secure Conversation	36
49	6.7.1	Overview.....	36
50	6.7.2	MessageChunk Structure	36
51	6.7.3	MessageChunks and Error Handling	40
52	6.7.4	Establishing a SecureChannel	40
53	6.7.5	Deriving Keys.....	42
54	6.7.6	Verifying Message Security.....	43
55	7	Transport Protocols	43
56	7.1	OPC UA TCP	43
57	7.1.1	Overview.....	43
58	7.1.2	Message Structure	44
59	7.1.3	Establishing a Connection.....	46
60	7.1.4	Closing a Connection	47
61	7.1.5	Error Handling	47
62	7.1.6	Error Recovery.....	48
63	7.2	SOAP/HTTP	49
64	7.2.1	Overview.....	49
65	7.2.2	XML Encoding.....	50
66	7.2.3	OPC UA Binary Encoding.....	51
67	7.3	HTTPS.....	51
68	7.3.1	Overview.....	51
69	7.3.2	XML Encoding.....	53
70	7.3.3	OPC UA Binary Encoding.....	53
71	7.4	Well Known Addresses	54
72	8	Normative Contracts	54
73	8.1	OPC Binary Schema	54
74	8.2	XML Schema and WSDL.....	54
75	Annex A (normative)	Constants	56
76	A.1	Attribute Ids	56
77	A.2	Status Codes	56
78	A.3	Numeric Node Ids	56
79	Annex B (normative)	OPC UA Nodeset.....	58
80	Annex C (normative)	Type Declarations for the OPC UA Native Mapping	59
81	Annex D (normative)	WSDL for the XML Mapping	60
82	D.1	XML Schema	60
83	D.2	WSDL Port Types	60
84	D.3	WSDL Bindings.....	60
85	Annex E (normative)	Security Settings Management	61
86	E.1	Overview	61
87	E.2	SecuredApplication	62
88	E.3	CertificateIdentifier	65
89	E.4	CertificateStoreIdentifier	66
90	E.5	CertificateList.....	67
91	E.6	CertificateValidationOptions.....	67
92	Annex F (normative)	Information Model XML Schema	

93	F.1 Overview	69
94	F.2 UANodeSet	69
95	F.3 UANode	70
96	F.4 Reference	70
97	F.5 UAType	70
98	F.6 UAIInstance	71
99	F.7 UAVariable	71
100	F.8 TranslationType	72
101	F.9 UADatatype	72
102	F.10 DataTypeDefinition	73
103	F.11 DataTypeField	73
104	F.12 Variant	74
105	F.13 Example (Informative)	74
106	Annex G (informative): Revision 1.02 Highlights	77
107		
108		

109		
110	Figure 1 – The OPC UA Stack Overview	5
111	Figure 2 – Encoding Integers in a Binary Stream.....	8
112	Figure 3 – Encoding Floating Points in a Binary Stream	9
113	Figure 4 – Encoding Strings in a Binary Stream.....	9
114	Figure 5 – Encoding GUIDs in a Binary Stream	10
115	Figure 6 – Encoding XmlElements in a Binary Stream	11
116	Figure 7 – A String NodeId	12
117	Figure 8 – A Two Byte NodeId.....	12
118	Figure 9 – A Four Byte NodeId	12
119	Figure 10 – Security Handshake.....	27
120	Figure 11 – Relevant XML Web Services Specifications	32
121	Figure 12 – The WS Secure Conversation Handshake	32
122	Figure 13 – OPC UA Secure Conversation MessageChunk	36
123	Figure 14 – OPC UA TCP Message Structure.....	46
124	Figure 15 – Establishing a OPC UA TCP Connection	47
125	Figure 16 – Closing a OPC UA TCP Connection.....	47
126	Figure 17 – Recovering an OPC UA TCP Connection.....	49
127	Figure 18 – Scenarios for the HTTPS Transport	52
128		
129		

130		
131	Table 1 – Built-in Data Types	6
132	Table 2 – Guid Structure	6
133	Table 3 – Supported Floating Point Types	9
134	Table 4 – NodeId Components	11
135	Table 5 – NodeId DataEncoding Values	11
136	Table 6 – Standard NodeId Binary DataEncoding	11
137	Table 7 – Two Byte NodeId Binary DataEncoding	12
138	Table 8 – Four Byte NodeId Binary DataEncoding	12
139	Table 9 – ExpandedNodeId Binary DataEncoding	13
140	Table 10 – DiagnosticInfo Binary DataEncoding	14
141	Table 11 – QualifiedName Binary DataEncoding	14
142	Table 12 – LocalizedText Binary DataEncoding	14
143	Table 13 – Extension Object Binary DataEncoding	15
144	Table 14 – Variant Binary DataEncoding	16
145	Table 15 – Data Value Binary DataEncoding	17
146	Table 16 – Sample OPC UA Binary Encoded Structure	18
147	Table 17 – XML Data Type Mappings for Integers	19
148	Table 18 – XML Data Type Mappings for Floating Points	19
149	Table 19 – Components of NodeId	21
150	Table 20 – Components of ExpandedNodeId	22
151	Table 21 – Components of Enumeration	25
152	Table 22 – SecurityPolicy	27
153	Table 23 – ApplicationInstanceCertificate	29
154	Table 24 – SignedSoftwareCertificate	30
155	Table 25 – Kerberos UserTokenPolicy	31
156	Table 26 – WS-* Namespace Prefixes	33
157	Table 27 – RST/SCT Mapping to an OpenSecureChannel Request	34
158	Table 28 – RSTR/SCT Mapping to an OpenSecureChannel Response	35
159	Table 29 – OPC UA Secure Conversation Message Header	37
160	Table 30 – Asymmetric Algorithm Security Header	38
161	Table 31 – Symmetric Algorithm Security Header	39
162	Table 32 – Sequence Header	39
163	Table 33 – OPC UA Secure Conversation Message Footer	39
164	Table 34 – OPC UA Secure Conversation Message Abort Body	40
165	Table 35 – OPC UA Secure Conversation OpenSecureChannel Service	41
166	Table 36 – Cryptography Key Generation Parameters	42
167	Table 37 – OPC UA TCP Message Header	44
168	Table 38 – OPC UA TCP Hello <i>Message</i>	44
169	Table 39 – OPC UA TCP Acknowledge Message	45
170	Table 40 – OPC UA TCP Error Message	45
171	Table 41 – OPC UA TCP Error Codes	48
172	Table 42 – WS-Addressing Headers	50

173 Table 43 – Well Known Addresses for Local Discovery Servers..... 54

174 Table A1 – Identifiers Assigned to Attributes 56

175 Table D1 – SecuredApplication 63

176 Table D2 – CertificateIdentifier 65

177 Table D3 – Structured Directory Store 66

178 Table D4 – CertificateStoreIdentifier 67

179 Table D5 – CertificateList 67

180 Table D6 – CertificateValidationOptions 68

181 Table E1 – UANodeSet 69

182 Table E2 – UANode..... 70

183 Table E3 – Reference 70

184 Table E4 – UAInstance..... 71

185 Table E5 – UAVariable 71

186 Table E6 – TranslationType..... 72

187 Table E7 – UADatatype 73

188 Table E8 – DataTypeDefinition 73

189 Table E9 – DataTypeField 74

190

191

192

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.

International Standard 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 standard is based on the following documents:

FDIS	Report on voting
65E/XX/FDIS	65E/XX/RVD

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

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

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.

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

- 245 • reconfirmed,
246 • withdrawn,
247 • replaced by a revised edition, or
248 • amended.

249

250 The National Committees are requested to note that for this publication the stability date is 2017.

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

253

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

OPC Unified Architecture 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*

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/>

62541-6/ED.2/CDV © IEC(E)

- 290 WS Trust: WS Trust 1.3
291 <http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html>
- 292 WS Secure Conversation: WS Secure Conversation 1.3
293 <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html>
- 294 WS Security Policy: WS Security Policy 1.2
295 <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html>
- 296 SSL/TLS: RFC 5246 - The TLS Protocol Version 1.2
297 <http://tools.ietf.org/html/rfc5246.txt>
- 298 X509: X.509 Public Key Certificate Infrastructure
299 <http://www.itu.int/rec/T-REC-X.509-200003-I/e>
- 300 WS-I Basic Profile 1.1: WS-I Basic Profile Version 1.1
301 <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>
- 302 WS-I Basic Security Profile 1.1: WS-I Basic Security Profile Version 1.1
303 <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html>
- 304 HTTP: RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1
305 <http://www.ietf.org/rfc/rfc2616.txt>
- 306 HTTPS: RFC 2818 - HTTP Over TLS
307 <http://www.ietf.org/rfc/rfc2818.txt>
- 308 Base64: RFC 3548 - The Base16, Base32, and Base64 Data Encodings
309 <http://www.ietf.org/rfc/rfc3548.txt>
- 310 X690 : ITU-T X.690 - Basic (BER), Canonical (CER) and Distinguished (DER) Encoding Rules
311 <http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>
- 312 X200 : ITU-T X.200 – Open Systems Interconnection – Basic Reference Model
313 <http://www.itu.int/rec/T-REC-X.200-199407-I/en>
- 314 IEEE-754: Standard for Binary Floating-Point Arithmetic
315 <http://grouper.ieee.org/groups/754/>
- 316 HMAC: HMAC - Keyed-Hashing for Message Authentication
317 <http://www.ietf.org/rfc/rfc2104.txt>
- 318 PKCS #1 : PKCS #1 - RSA Cryptography Specifications Version 2.0
319 <http://www.ietf.org/rfc/rfc2437.txt>
- 320 PKCS #12 : PKCS 12 v1.0: Personal Information Exchange Syntax
321 <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/pkcs-12v1.pdf>
322
- 323 FIPS 180-2: Secure Hash Standard (SHA)
324 <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>
325
- 326 FIPS 197: Advanced Encryption Standard (AES)
327 <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
328
- 329 UTF8: UTF-8, a transformation format of ISO 10646
330 <http://tools.ietf.org/html/rfc3629>

331 RFC 3280: RFC 3280 - X.509 Public Key Infrastructure Certificate and CRL Profile

332 <http://www.ietf.org/rfc/rfc3280.txt>

333 RFC 4514: RFC 4514 - LDAP: String Representation of Distinguished Names

334 <http://www.ietf.org/rfc/rfc4514.txt>

335 NTP: RFC 1305 - Network Time Protocol (Version 3)

336 <http://www.ietf.org/rfc/rfc1305.txt>

337 Kerberos: WS Security Kerberos Token Profile 1.1

338 <http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf>

339 **3 Terms, definitions and conventions**

340 **3.1 Terms and definitions**

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

343 **3.1.1**

344 **Data Encoding**

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

346 **3.1.2**

347 **Mapping**

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

349 NOTE For example, the OPC UA Binary Encoding is a *Mapping* that specifies how to serialize OPC UA data structures as
350 sequences of bytes.

351 **3.1.3**

352 **Security Protocol**

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

354 **3.1.4**

355 **Stack Profile**

356 a combination of *DataEncodings*, *SecurityProtocol* and *TransportProtocol Mappings*

357 NOTE OPC UA applications implement one or more *StackProfiles* and can only communicate with OPC UA applications
358 that support a *StackProfile* that they support.

359 **3.1.5**

360 **Transport Protocol**

361 a way to exchange serialized OPC UA *Messages* between OPC UA applications

3.2 Abbreviations and symbols

API	Application Programming Interface
ASN.1	Abstract Syntax Notation #1 (used in X690)
BP	WS-I Basic Profile Version
BSP	WS-I Basic Security Profile
CSV	Comma Separated Value (File Format)
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IPSec	Internet Protocol Security
RST	Request Security Token
OID	Object Identifier (used with ASN.1)
RSTR	Request Security Token Response
SCT	Security Context Token
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)
UTF8	Unicode Transformation Format (8-bit) (Defined in UTF8)
UA	Unified Architecture
UASC	OPC UA Secure Conversation
WS-*	XML Web Services Specifications
WSS	WS Security
WS-SC	WS Secure Conversation
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 6 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.

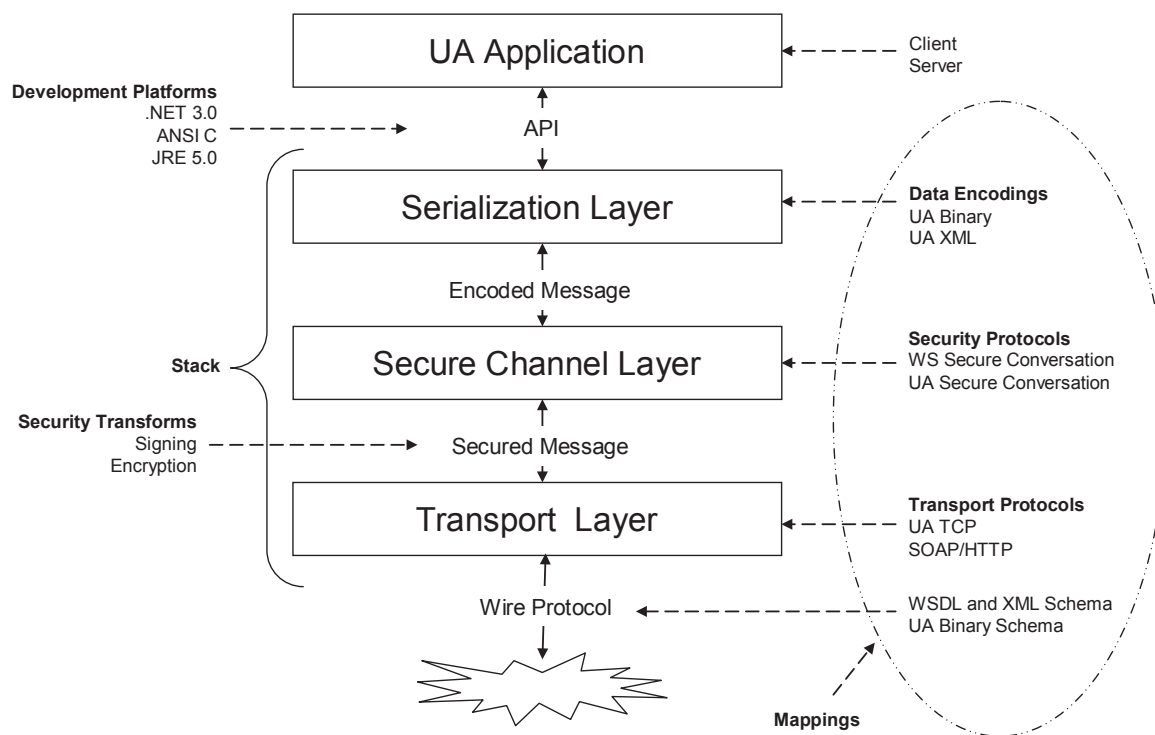


Figure 1 – The OPC UA Stack Overview

The layers described in this specification do not correspond to layers in the OSI 7 layer model [X200]. 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 two data encodings: OPC UA Binary and OPC UA XML. 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.

433

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.
3	Byte	An integer value between 0 and 256.
4	Int16	An integer value between -32 768 and 32 767.
5	UInt16	An integer value between 0 and 65 535.
6	Int32	An integer value between -2 147 483 648 and 2 147 483 647.
7	UInt32	An integer value between 0 and 429 4967 295.
8	Int64	An integer value between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807
9	UInt64	An integer value between 0 and 18 446 744 073 709 551 615.
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 a 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.

434

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

438 5.1.3 Guid

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

440

Table 2 – Guid Structure

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

441

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

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

444 Where Data1 is 8 characters wide, Data2 and Data3 are 4 characters wide and each *Byte* in Data4 is
 445 2 characters wide. Each value is formatted as a hexadecimal number padded zeros. A typical *Guid*
 446 value would look like this when formatted as a string:

447 `C496578A-0DFE-4b8f-870A-745238C6AEAE`

448

449 5.1.4 ByteString

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

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

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

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

462 5.1.5 ExtensionObject`

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

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

472 *Server* implementers should use namespace qualified numeric *NodeIds* for any *DataTypeEncoding*
473 *Objects* they define. This will minimize the overhead introduced by packing *Complex Data* values
474 into *ExtensionObjects*.

475 5.1.6 Variant

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

479 *Variants* can be empty. An empty *Variant* is described as having a null value and should be treated
480 like a null column in a SQL database. A null value in a *Variant* may not be the same as a null value
481 for data types that support nulls such as *Strings*. Some *Development Platforms* may not be able to
482 preserve the distinction between a null for a *DataType* and a null for a *Variant*. Therefore
483 *Applications* shall not rely on this distinction.

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

485 *DataValue* and *DiagnosticInfo* types only have meaning when returned in a response message with
486 an associated *StatusCode*. As a result, *Variants* cannot contain instances of *DataValue* or
487 *DiagnosticInfo*.

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

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 *Complex Data* structure it represents. This allows a decoder to skip over types that it does not recognize.

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.

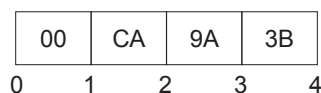


Figure 2 – Encoding Integers in a Binary Stream

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 3 lists the bit ranges for the supported floating point types.

Table 3 – 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.

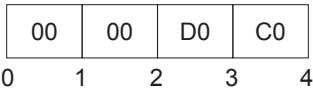


Figure 3 – Encoding Floating Points in a Binary Stream

5.2.2.4 String

All *String* values are encoded as a sequence of UTF8 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.

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



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:

- 551 a) A date/time value is encoded as 0 if either
- 552 1) The value is equal to or earlier than 1601-01-01 12:00AM.
- 553 2) The value is the earliest date that can be represented with the *DevelopmentPlatform's*
- 554 encoding.
- 555 b) A date/time is encoded as the maximum value for an *Int64* if either
- 556 1) The value is equal to or greater than 9999-01-01 11:59:59PM,
- 557 2) The value is the latest date that can be represented with the *DevelopmentPlatform's*
- 558 encoding.
- 559 c) A date/time is decoded as the earliest time that can be represented on the platform if either
- 560 1) The encoded value is 0,
- 561 2) The encoded value represents a time earlier than the earliest time that can be
- 562 represented with the *DevelopmentPlatform's* encoding.
- 563 d) A date/time is decoded as the latest time that can be represented on the platform if either
- 564 1) The encoded value is the maximum value for an *Int64*,
- 565 2) The encoded value represents a time later than the latest time that can be
- 566 represented with the *DevelopmentPlatform's* encoding.
- 567 These rules imply that the earliest and latest times that can be represented on a given platform are
- 568 invalid date/time values and should be treated that way by *Applications*.

569 A decoder shall truncate the value if a decoder encounters a *DateTime* value with a resolution that is

570 greater than the resolution supported on the *DevelopmentPlatform*.

571 5.2.2.6 Guid

572 A *Guid* is encoded in a structure as shown in Table 2. Fields are encoded sequentially according to

573 the data type for field.

574 Figure 5 illustrates how the *Guid* "72962B91-FA75-4ae6-8D28-B404DC7DAF63" should be encoded

575 in a byte stream.

Data1				Data2		Data3		Data4								
91	2B	96	72	75	FA	E6	4A	8D	28	B4	04	DC	7D	AF	63	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

576

577 **Figure 5 – Encoding Guids in a Binary Stream**

578

579 5.2.2.7 ByteString

580 A *ByteString* is encoded as sequence of bytes preceded by its length in bytes. The length is encoded

581 as a 32-bit signed integer as described above.

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

583 5.2.2.8 XmlElement

584 An *XmlElement* is an XML fragment serialized as UTF8 string and then encoded as *ByteString*.

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

Length				<A>			Hot			水							
0D	00	00	00	3C	41	3E	72	6F	74	E6	B0	B4	3C	3F	41	3E	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 6 – Encoding XmlElements in a Binary Stream

5.2.2.9 NodeId

The components of a *NodeId* are described the Table 4.

Table 4 – NodeId Components

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

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 5. The tables that follow describe the structure of each possible format (they exclude the byte which indicates the format).

Table 5 – 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 <i>String</i> value.
Guid	0x04	A <i>Guid</i> value.
ByteString	0x05	An opaque (<i>ByteString</i>) value.
NamespaceUri Flag	0x80	See discussion of <i>ExpandedNodeId</i> in 5.2.2.10.
ServerIndex Flag	0x40	See discussion of <i>ExpandedNodeId</i> in 5.2.2.10.

The standard *NodeId DataEncoding* has the structure shown in Table 6. The standard *DataEncoding* is used for all formats that do not have an explicit format defined.

Table 6 – Standard NodeId Binary DataEncoding

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

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

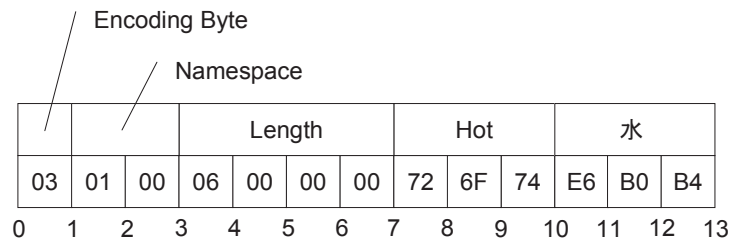


Figure 7 – A String NodeId

605 The Two Byte *NodeId DataEncoding* has the structure shown in Table 7.

Table 7 – Two Byte NodeId Binary DataEncoding

Name	Data Type	Description
Identifier	Byte	The <i>Namespace</i> is the default OPC UA namespace (i.e. 0). The <i>Identifier</i> Type is 'Numeric'. The <i>Identifier</i> shall be in the range 0 to 255.

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

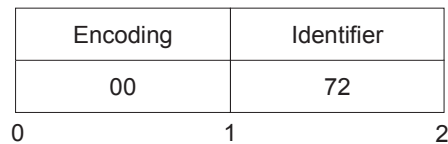


Figure 8 – A Two Byte NodeId

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

Table 8 – Four Byte NodeId Binary DataEncoding

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

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

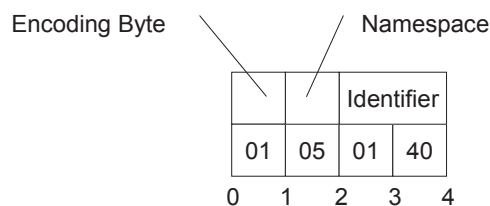


Figure 9 – A Four Byte NodeId

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 9.

Table 9 – ExpandedNodeId Binary DataEncoding

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

5.2.2.11 StatusCode

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

5.2.2.12 DiagnosticInfo

A *DiagnosticInfo* structure is described in IEC 62541-4, 7.8. It specifies a number of fields that could be missing. For that reason, the encoding uses a bit mask to indicate which fields are actually present in the encoded form.

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

647

Table 10 – 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.
LocalizedText	Int32	A human readable summary of the status code.
Locale	Int32	The locale used for the localized text.
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.

648

649 5.2.2.13 QualifiedName650 A *QualifiedName* structure is encoded as shown in Table 11.651 The abstract *QualifiedName* structure is defined in IEC 62541-3, 8.3.

652

Table 11 – QualifiedName Binary DataEncoding

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

653

654 5.2.2.14 LocalizedText655 A *LocalizedText* structure contains two fields that could be missing. For that reason, the encoding
656 uses a bit mask to indicate which fields are actually present in the encoded form.657 The abstract *LocalizedText* structure is defined in IEC 62541-3, 8.5.

658

Table 12 – 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.

659

660 5.2.2.15 ExtensionObject661 An *ExtensionObject* is encoded as sequence of bytes prefixed by the *NodeId* of its
662 *DataEncoding* and the number of bytes encoded.663 An *ExtensionObject* may be encoded by the *Application* which means it is passed as a *ByteStream* of
664 an *XmlElement* to the encoder. In this case, the encoder will be able to write the number of bytes in

Documents
 Document Sharing Hub

the object before it encodes the bytes. However, an *ExtensionObject* may know how to encode/decode itself which means the encoder shall calculate the number of bytes before it encodes the object or it shall be able to seek backwards in the stream and update the length after encoding the body.

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

The serialized form of an *ExtensionObject* is shown in Table 13.

Table 13 – Extension Object Binary DataEncoding

Name	Data Type	Description
TypeId	NodeId	The identifier for the <i>DataTypeEncoding</i> node in the <i>Server's AddressSpace</i> . <i>ExtensionObjects</i> defined by the OPC UA specification have a numeric node identifier assigned to them with a <i>NamespaceIndex</i> of 0. The numeric identifiers are defined in A.1.
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 <i>ByteString</i> . 0x02 The body is encoded as a <i>XmlElement</i> .
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 <i>ByteString</i> bodies. For <i>XmlElement</i> bodies this field contains the XML encoded as a UTF-8 string without any null terminator.

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

5.2.2.16 Variant

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

The structure of a *Variant* is shown in Table 14.