DEUTSCHE NORM

August 2021

DIN EN IEC 62541-9

ICS 25.040.40; 35.200; 35.240.50

Ersatz für
DIN EN 62541-9:2015-11
Siehe Anwendungsbeginn

OPC Unified Architecture – Teil 9: Alarme und Zustände (IEC 62541-9:2020); Englische Fassung EN IEC 62541-9:2020

OPC Unified Architecture –
Part 9: Alarms and Conditions
(IEC 62541-9:2020);
English version EN IEC 62541-9:2020

Architecture unifiée OPC –
Partie 9: Alarmes et Conditions
(IEC 62541-9:2020);
Version anglaise EN IEC 62541-9:2020

Gesamtumfang 147 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2020-07-23 angenommene Europäische Norm als DIN-Norm ist 2021-08-01.

Für DIN EN 62541-9:2015-11 besteht eine Übergangsfrist bis 2023-07-23.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN EN IEC 62541-9:2019-09.

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

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

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

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

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

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

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

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

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

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

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

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

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

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

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

Da sich die Benutzer der vorliegenden Norm der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN IEC 62541-9 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt und Abschnitt 3 (außer Unterabschnitt 3.3), der die Begriffe und Abkürzungen festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA der vorliegenden Norm hinzugefügt. Für die meisten der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Diese Norm steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

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

Änderungen

Gegenüber DIN EN 62541-9:2015-11 wurden folgende Änderungen vorgenommen:

a) optionale Engineering-Einheiten zur Definition von RateOfChange-Alarmen hinzugefügt;

b) zur Erfüllung des IEC-62682-Modells wurden die folgenden Elemente hinzugefügt:
   - AlarmBedingungsTypen-Zustände: Unterdrückung, Stille, AußerBetrieb, eingerastet,
   - AlarmBedingungsTypen-Eigenschaften: VerzögerungAn, VerzögerungAus, GruppenErster, ErneuterAlarmZeit,
   - Neue Alarmtypen: DiskrepanzAlarm, AbweichungsAlarm, GeräteDiagnoseAlarm, SystemDiagnoseAlarm;

c) Anhang hinzugefügt, der festlegt, wie die Konzepte dieses OPC-UA-Teils auf IEC 62682 und ISA 18.2 abgebildet werden;

d) neue BedingungsKlassen hinzugefügt: Sicherheit, IntensivVerwaltet, Statistisch, Prüfung, Schulung;

e) AlarmTyp ZertifikatsAblauf hinzugefügt;

f) Alarmmetrikenmodell hinzugefügt.

Frühere Ausgaben

DIN EN 62541-9: 2013-06, 2015-11

AVA Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Nationaler Anhang NA
(informativ)

Auszugsweise Übersetzung

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

1 Anwendungsbereich

Dieser Teil der Normenreihe IEC 62541 legt die Darstellung von *Alarmen* und *Zuständen* in der OPC Unified Architecture (vereinheitlichte Architektur der plattformunabhängigen Kommunikation (OPC-UA)) fest. Darin eingeschlossen ist die Darstellung von *Alarmen* und *Zuständen* mittels des *InformationsModells* (en: *Information Model*) im Adressraum des OPC-UA. Andere Aspekte von Alarmsystemen, wie etwa Alarmphilosophie, Lebenszyklus, Alarmreaktionszeiten, Alarmtypen und viele andere Einzelheiten, werden in Dokumenten wie IEC 62682 und ISA 18.2 behandelt. Das *Alarm-* und *Zustände-Informationsmodell* in dieser Spezifikation wird in Übereinstimmung mit IEC 62682 und ISA 18.2 entworfen.

3 Begriffe, Abkürzungen und verwendete Datentypen

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC TR 62541-1, IEC 62541-3, IEC 62541-4, IEC 62541-5 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
Quittieren
(en: Acknowledge)
Handlung eines *Operators* (en: *Operator*), die das Erkennen eines *Alarms* anzeigt

Anmerkung 1 zum Begriff: Diese Definition wurde aus EEMUA übernommen. Der Begriff „Akzeptieren“ (en: Accept) ist ein weiterer gängiger Begriff für Quittieren. Sie können synonym verwendet werden. Dieses Dokument verwendet Quittieren.

3.1.2
aktiv
(en: Active)
Zustand eines Alarms, der anzeigt, dass die Situation, die der *Alarm* repräsentiert, aktuell existiert

Anmerkung 1 zum Begriff: Andere gängige Begriffe, definiert nach EEMUA, sind „anstehend“ (en: Standing) für einen *aktiven Alarm* und „nullgestellt“ (en: Cleared), wenn der *Zustand* (en: *Condition*) wieder bestimmungsgemäß und nicht länger *aktiv* ist.

3.1.3
AdaptiverAlarm
(en: AdaptiveAlarm)
Alarm, für den der Einstellwert und die Grenzen von einem Algorithmus geändert werden

Anmerkung 1 zum Begriff: *AdaptiveAlarme* sind Alarme, die durch Algorithmen automatisch angepasst werden. Diese Algorithmen können Zustände in einer Anlage erfassen und Einstellwerte oder Grenzen ändern, um zu verhindern, dass

AVA Documents Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Alarme aktiviert werden. Diese Änderungen werden in vielen Fällen ohne ein Eingreifen des *Operators* (en: *Operator*) vorgenommen.

3.1.4
AlarmFlut
(en: AlarmFlood)
Zustand, in dessen Rahmen die Alarmrate höher ist als der *Operator* sie effektiv handhaben kann

Anmerkung 1 zum Begriff: OPC-UA legt die Bedingungen, die als Alarmflut betrachtet würden, nicht fest; diese Bedingungen werden in anderen Spezifikationen, wie z. B. IEC 62682 oder ISA 18.2, festgelegt.

3.1.5
AlarmUnterdrückungsGruppe
(en: AlarmSuppressionGroup)
Gruppe von *Alarmen*, welche zur Unterdrückung anderer *Alarme* verwendet wird

Anmerkung 1 zum Begriff: Eine *AlarmUnterdrückungsGruppe* ist eine Instanz eines *AlarmGruppenTyps* (en: *AlarmGroupType*), der zur Unterdrückung anderer Alarme verwendet wird. Wenn ein beliebiger Alarm in der Gruppe aktiv ist, ist auch die *AlarmUnterdrückungsGruppe* aktiv. Wenn alle Alarme in der *AlarmUnterdrückungsGruppe* inaktiv sind, ist auch die *AlarmUnterdrückungsGruppe* inaktiv.

Anmerkung 2 zum Begriff: Der betroffene Alarm verweist auf *AlarmUnterdrückungsGruppen* durch den *ReferenzTyp* (en: *ReferenzType*) *BesitztAlarmUnterdrückungsGruppe* (en: *HasAlarmSuppressionGroup*).

3.1.6
ZustandsKlasse
(en: ConditionClass)
Gruppierung von *Zuständen* (en: *Condition*), die anzeigt, in welcher Domäne oder für welchen Zweck ein bestimmter *Zustand* verwendet wird

Anmerkung 1 zum Begriff: Einige oberste *ZustandsKlassen* (en: top-level *ConditionClasses*) sind in dieser Spezifikation definiert. Anbieter oder Organisationen können davon konkretere Klassen ableiten oder andere oberste Klassen definieren.

3.1.7
ZustandsZweig
(en: ConditionBranch)
bestimmter Status eines Zustands

Anmerkung 1 zum Begriff: Der Server kann *ZustandsZweige* sowohl für aktuelle Status als auch für vorherige Status verwalten.

3.1.8
ZustandsQuelle
(en: ConditionSource)
Element, auf dem ein bestimmter *Zustand* (en: *Condition*), basiert oder auf das er bezogen ist

Anmerkung 1 zum Begriff: Üblicherweise ist dies eine Variable (en: Variable), welche einen Prozesswert (z. B. FIC101) repräsentiert, oder ein *Objekt*, welches ein Gerät oder Teilsystem repräsentiert.

Anmerkung 2 zum Begriff: Bei *Ereignissen* (en: *Events*), die für *Zustände* (en: *Condition*) erzeugt werden, enthält das *QuellKnoten Merkmal* (en: *SourceNode Property*) (geerbt vom *BasisEreignisTyp* (en: *BaseEventType*)) die *KnotenID* (en: *NodeId*) der *ZustandsQuelle*.

3.1.9
Bestätigen
(en: Confirm)
Handlung eines *Operators*, die den *Server* informiert, dass eine korrigierende Handlung ausgeführt wurde, welche die Ursache des *Alarms* betrifft

AVA Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

3.1.10
Deaktivieren
(en: Disable)
Handlung, die ein System so konfiguriert, dass der *Alarm* nicht erzeugt wird, auch wenn die *AlarmBedingung* (en: *Alarm Condition*) erfüllt ist

Anmerkung 1 zum Begriff: Diese Definition wurde von EEMUA übernommen und ist in EEMUA weiterführend festgelegt.

Anmerkung 2 zum Begriff: In IEC 62682 wird auf „deaktivieren“ (en: disable) als „Außer Betrieb“ (en: Out of Service) verwiesen.

3.1.11
Einrastender Alarm
(en: LatchingAlarm)
Alarm, der im Alarmzustand verbleibt, nachdem der Prozesszustand in den Normalzustand zurückgekehrt ist, und der ein Zurücksetzen durch den *Operator* erforderlich macht, bevor der Alarm in den Normalzustand zurückkehrt

Anmerkung 1 zum Begriff: Bei einrastenden Alarmen handelt es sich üblicherweise um Diskrepanzalarme, bei denen eine Handlung nicht innerhalb einer bestimmten Zeitdauer vorgenommen wird. Sobald die Handlung vorgenommen wurde, bleibt der Alarm aktiv, bis er zurückgesetzt wird.

3.1.12
Operator
(en: Operator)
besonderer Benutzer, der beauftragt ist, einen Teil eines Prozesses zu beobachten und zu steuern

Anmerkung 1 zum Begriff: „Ein Mitglied des Operator-Teams, welches beauftragt wurde, einen Teil des Prozesses zu beobachten und zu steuern, und der an einer Leitsystemkonsole arbeitet,“ wie nach EEMUA festgelegt. Im vorliegenden Dokument ist ein Operator ein besonderer Benutzer. Alle Festlegungen, die auf einen allgemeinen Benutzer angewendet werden, gelten auch für Operatoren.

3.1.13
Aktualisieren
(en: Refresh)
Handlung der Bereitstellung einer Erneuerung für ein *EreignisAbonnement* (en: *EventSubscription*), die alle *Alarme* bereitstellt, welche als *Erhalten* (en: *Retained*) betrachtet werden

Anmerkung 1 zum Begriff: Dieses Konzept ist genauer in EEMUA beschrieben.

3.1.14
Erhalten
(en: Retain)
Alarm, der in einem Zustand ist, der für einen *Client* interessant ist, welcher seinen Status des *Zustands* (en: *Conditions*) mit dem Zustand im *Servers* synchronisieren will

3.1.15
Zurückstellen
(en: Shelving)
Einrichtung, mit der der *Operator* in der Lage ist, temporär einen *Alarm* daran zu hindern, dem *Operator* angezeigt zu werden, wenn dies den *Operator* ablenkt

Anmerkung 1 zum Begriff: „Ein zurückgestellter Alarm wird von der Liste entfernt und wird nicht wieder angezeigt, bis die Zurückstellung aufgehoben wird. Diese Definition wurde von EEMUA übernommen.

3.1.16
Unterdrücken
(en: Suppress)
Handlung zum Ermitteln, ob ein Alarm nicht auftreten sollte

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Anmerkung 1 zum Begriff: „Ein Alarm ist unterdrückt, wenn logische Kriterien angewendet werden, um zu bestimmen, dass der Alarm nicht auftreten sollte, selbst wenn die *AlarmBedingung* (z. B. Alarm-Einstellungen überschritten) erfüllt ist“ wie nach EEMUA definiert. In IEC 62682 werden unterdrückte Alarme auch als „nach Auslegung unterdrückt“ beschrieben, d. h. dass das System mit der Logik ausgelegt wurde, einen Alarm zu unterdrücken, wenn bestimmte Kriterien erfüllt sind. Wenn z. B. eine Verarbeitungseinheit abgeschaltet wird, werden Alarme geringerer Priorität für alle Ausrüstungsteile der abgeschalteten Einheit unterdrückt.

3.2 Abkürzungen

A&E	Alarme & Ereignisse (wie verwendet für OPC COM) (en: Alarm & Event (as used for OPC COM))
COM	(Microsoft Windows) Komponenten Objekt Model (en: (Microsoft Windows) Component Object Model)
DA	Datenzugriff (en: data access)
UA	vereinheitlichte Architektur (en: Unified Architecture)

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

– Leerseite –

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN IEC 62541-9

August 2020

ICS 35.100.05; 25.040.40

Supersedes EN 62541-9:2015 and all of its amendments and corrigenda (if any)

English Version

OPC Unified Architecture - Part 9: Alarms and Conditions (IEC 62541-9:2020)

Architecture unifiée OPC - Partie 9: Alarmes et Conditions (IEC 62541-9:2020)

OPC Unified Architecture - Teil 9: Alarme und Zustände (IEC 62541-9:2020)

This European Standard was approved by CENELEC on 2020-07-23. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

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

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

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

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

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

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

European foreword

The text of document 65E/709/FDIS, future edition 3 of IEC 62541-9, prepared by SC 65E "Devices and integration in enterprise systems" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN IEC 62541-9:2020.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement (dop) 2021-04-23
- latest date by which the national standards conflicting with the document have to be withdrawn (dow) 2023-07-23

This document supersedes EN 62541-9:2015 and all of its amendments and corrigenda (if any).

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

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

Endorsement notice

The text of the International Standard IEC 62541-9:2020 was approved by CENELEC as a European Standard without any modification.

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

Publication	Year	Title	EN/HD	Year
IEC/TR 62541-1	–	OPC unified architecture – Part 1: Overview and concepts	CLC/TR 62541-1	–
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 62541-6	–	OPC Unified Architecture – Part 6: Mappings	–	–
IEC 62541-7	–	OPC unified architecture – Part 7: Profiles	–	–
IEC 62541-8	–	OPC Unified Architecture – Part 8: Data Access	–	–
IEC 62541-11	–	OPC Unified Architecture – Part 11: Historical Access	–	–
IEC 62682	–	Management of alarms systems for the process industries	–	–
EEMUA 191	–	Alarm systems – A guide to design, management and procurement	–	–

AVA Documents
Document Sharing Hub

CONTENTS

European foreword 10
Annex ZA (normative) Normative references to international publications with their corresponding European publications 11
1 Scope 21
2 Normative references 21
3 Terms, definitions, abbreviated terms and data types used 21
3.1 Terms and definitions 21
3.2 Abbreviated terms 23
3.3 Data types used 24
4 Concepts 24
4.1 General 24
4.2 Conditions 24
4.3 Acknowledgeable Conditions 26
4.4 Previous states of Conditions 27
4.5 Condition state synchronization 28
4.6 Severity, quality, and comment 28
4.7 Dialogs 29
4.8 Alarms 29
4.9 Multiple active states 30
4.10 Condition instances in the AddressSpace 31
4.11 Alarm and Condition auditing 32
5 Model 32
5.1 General 32
5.2 Two-state state machines 33
5.3 ConditionVariable 35
5.4 ReferenceTypes 35
5.4.1 General 35
5.4.2 HasTrueSubState ReferenceType 35
5.4.3 HasFalseSubState ReferenceType 36
5.4.4 HasAlarmSuppressionGroup ReferenceType 36
5.4.5 AlarmGroupMember ReferenceType 37
5.5 Condition Model 37
5.5.1 General 37
5.5.2 ConditionType 38
5.5.3 Condition and branch instances 41
5.5.4 Disable Method 42
5.5.5 Enable Method 42
5.5.6 AddComment Method 43

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

5.5.7 ConditionRefresh Method 44
5.5.8 ConditionRefresh2 Method 46
5.6 Dialog Model 48
5.6.1 General 48
5.6.2 DialogConditionType 48
5.6.3 Respond Method 50
5.7 Acknowledgeable Condition Model 50
5.7.1 General 50
5.7.2 AcknowledgeableConditionType 51
5.7.3 Acknowledge Method 52
5.7.4 Confirm Method 53
5.8 Alarm model 54
5.8.1 General 54
5.8.2 AlarmConditionType 55
5.8.3 AlarmGroupType 60
5.8.4 Reset Method 60
5.8.5 Silence Method 61
5.8.6 Suppress Method 62
5.8.7 Unsuppress Method 63
5.8.8 RemoveFromService Method 64
5.8.9 PlaceInService Method 65
5.8.10 ShelvedStateMachineType 66
5.8.11 LimitAlarmType 71
5.8.12 Exclusive limit types 73
5.8.13 NonExclusiveLimitAlarmType 76
5.8.14 Level Alarm 78
5.8.15 Deviation Alarm 79
5.8.16 Rate of change Alarms 81
5.8.17 Discrete Alarms 82
5.8.18 DiscrepancyAlarmType 85
5.9 ConditionClasses 86
5.9.1 Overview 86
5.9.2 BaseConditionClassType 86
5.9.3 ProcessConditionClassType 87
5.9.4 MaintenanceConditionClassType 87
5.9.5 SystemConditionClassType 87
5.9.6 SafetyConditionClassType 88
5.9.7 HighlyManagedAlarmConditionClassType 88
5.9.8 TrainingConditionClassType 89

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

5.9.9 StatisticalConditionClassType 89
5.9.10 TestingConditionSubClassType 89
5.10 Audit Events 90
5.10.1 Overview 90
5.10.2 AuditConditionEventType 90
5.10.3 AuditConditionEnableEventType 91
5.10.4 AuditConditionCommentEventType 91
5.10.5 AuditConditionRespondEventType 92
5.10.6 AuditConditionAcknowledgeEventType 92
5.10.7 AuditConditionConfirmEventType 92
5.10.8 AuditConditionShelvingEventType 93
5.10.9 AuditConditionSuppressionEventType 93
5.10.10 AuditConditionSilenceEventType 94
5.10.11 AuditConditionResetEventType 94
5.10.12 AuditConditionOutOfServiceEventType 94
5.11 Condition Refresh related Events 95
5.11.1 Overview 95
5.11.2 RefreshStartEventType 95
5.11.3 RefreshEndEventType 95
5.11.4 RefreshRequiredEventType 96
5.12 HasCondition Reference type 96
5.13 Alarm and Condition status codes 97
5.14 Expected A&C server behaviours 97
5.14.1 General 97
5.14.2 Communication problems 98
5.14.3 Redundant A&C servers 98
6 AddressSpace organisation 98
6.1 General 98
6.2 EventNotifier and source hierarchy 98
6.3 Adding Conditions to the hierarchy 99
6.4 Conditions in InstanceDeclarations 100
6.5 Conditions in a VariableType 101
7 System State and alarms 101
7.1 Overview 101
7.2 HasEffectDisable 102
7.3 HasEffectEnable 102
7.4 HasEffectSuppress 103
7.5 HasEffectUnsuppressed 103
8 Alarm metrics 104

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

8.1 Overview 104
8.2 AlarmMetricsType 104
8.3 AlarmRateVariableType 105
8.4 Reset Method 106
Annex A (informative) Recommended localized names 107
A.1 Recommended state names for TwoState variables 107
A.1.1 LocaleId "en" 107
A.1.2 LocaleId "de" 107
A.1.3 LocaleId "fr" 108
A.2 Recommended dialog response options 110
Annex B (informative) Examples 111
B.1 Examples for Event sequences from Condition instances 111
B.1.1 Overview 111
B.1.2 Server maintains current state only 111
B.1.3 Server maintains previous states 112
B.2 AddressSpace examples 114
Annex C (informative) Mapping to EEMUA 117
Annex D (informative) Mapping from OPC A&E to OPC UA A&C 119
D.1 Overview 119
D.2 Alarms and Events COM UA wrapper 119
D.2.1 Event Areas 119
D.2.2 Event sources 119
D.2.3 Event categories 120
D.2.4 Event attributes 121
D.2.5 Event subscriptions 121
D.2.6 Condition instances 124
D.2.7 Condition Refresh 124
D.3 Alarms and Events COM UA proxy 124
D.3.1 General 124
D.3.2 Server status mapping 125
D.3.3 Event Type mapping 125
D.3.4 Event category mapping 125
D.3.5 Event Category attribute mapping 126
D.3.6 Event Condition mapping 130
D.3.7 Browse mapping 130
D.3.8 Qualified names 131
D.3.8.1 Qualified name syntax 131
D.3.8.2 Namespace table 131
D.3.9 Subscription filters 132

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

D.3.9.1 General 132
D.3.9.2 Filter by Event, category or severity 132
D.3.9.3 Filter by source 132
D.3.9.4 Filter by area 132
Annex E (informative) IEC 62682 Mapping 134
E.1 Overview 134
E.2 Terms 134
E.3 Alarm records and State indications 141
Annex F (informative) System State 142
F.1 Overview 142
F.2 SystemStateStateMachineType 143
Bibliography 147

Figure 1 – Base Condition state model 25
Figure 2 – AcknowledgeableConditions state model 26
Figure 3 – Acknowledge state model 27
Figure 4 – Confirmed Acknowledge state model 27
Figure 5 – Alarm state machine model 29
Figure 6 – Typical Alarm Timeline example 30
Figure 7 – Multiple active states example 31
Figure 8 – ConditionType hierarchy 33
Figure 9 – Condition model 38
Figure 10 – DialogConditionType overview 48
Figure 11 – AcknowledgeableConditionType overview 51
Figure 12 – AlarmConditionType Hierarchy Model 55
Figure 13 – Alarm Model 56
Figure 14 – Shelve state transitions 66
Figure 15 – ShelvedStateMachineType model 67
Figure 16 – LimitAlarmType 71
Figure 17 – ExclusiveLimitStateMachineType 73
Figure 18 – ExclusiveLimitAlarmType 75
Figure 19 – NonExclusiveLimitAlarmType 77
Figure 20 – DiscreteAlarmType Hierarchy 82
Figure 21 – ConditionClass type hierarchy 86
Figure 22 – AuditEvent hierarchy 90
Figure 23 – Refresh Related Event Hierarchy 95
Figure 24 – Typical HasNotifier Hierarchy 99
Figure 25 – Use of HasCondition in a HasNotifier hierarchy 100
Figure 26 – Use of HasCondition in an InstanceDeclaration 101

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Figure 27 – Use of HasCondition in a VariableType 101
Figure B.1 – Single state example 111
Figure B.2 – Previous state example 113
Figure B.3 – HasCondition used with Condition instances 115
Figure B.4 – HasCondition reference to a Condition type 116
Figure B.5 – HasCondition used with an instance declaration 116
Figure D.1 – The type model of a wrapped COM A&E server 121
Figure D.2 – Mapping UA Event Types to COM A&E Event Types 125
Figure D.3 – Example mapping of UA Event Types to COM A&E categories 126
Figure D.4 – Example mapping of UA Event Types to A&E categories with attributes 130
Figure F.1 – SystemState transitions 143
Figure F.2 – SystemStateStateMachineType Model 144

Table 1 – Parameter types defined in IEC 62541-3 24
Table 2 – Parameter types defined in IEC 62541-4 24
Table 3 – TwoStateVariableType definition 34
Table 4 – ConditionVariableType definition 35
Table 5 – HasTrueSubState ReferenceType 36
Table 7 – HasAlarmSuppressionGroup ReferenceType 37
Table 8 – AlarmGroupMember ReferenceType 37
Table 9 – ConditionType definition 38
Table 10 – SimpleAttributeOperand 41
Table 11 – Disable result codes 42
Table 12 – Disable Method AddressSpace definition 42
Table 13 – Enable result codes 43
Table 14 – Enable Method AddressSpace definition 43
Table 15 – AddComment arguments 43
Table 16 – AddComment result codes 44
Table 17 – AddComment Method AddressSpace definition 44
Table 18 – ConditionRefresh parameters 45
Table 19 – ConditionRefresh result codes 45
Table 20 – ConditionRefresh Method AddressSpace definition 46
Table 21 – ConditionRefresh2 parameters 46
Table 22 – ConditionRefresh2 result codes 47
Table 23 – ConditionRefresh2 Method AddressSpace definition 48
Table 24 – DialogConditionType definition 49
Table 25 – Respond parameters 50
Table 26 – Respond Result Codes 50
Table 27 – Respond Method AddressSpace definition 50

AVA Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Table 28 – AcknowledgeableConditionType definition 51
Table 29 – Acknowledge parameters 52
Table 30 – Acknowledge result codes 52
Table 31 – Acknowledge Method AddressSpace definition 53
Table 32 – Confirm Method parameters 53
Table 33 – Confirm result codes 54
Table 34 – Confirm Method AddressSpace definition 54
Table 35 – AlarmConditionType definition 57
Table 36 – AlarmGroupType definition 60
Table 37 – Silence result codes 61
Table 38 – Reset Method AddressSpace definition 61
Table 39 – Silence result codes 61
Table 40 – Silence Method AddressSpace definition 62
Table 41 – Suppress result codes 62
Table 42 – Suppress Method AddressSpace definition 63
Table 43 – Unsuppress result codes 63
Table 44 – Unsuppress Method AddressSpace definition 64
Table 45 – RemoveFromService result codes 64
Table 46 – RemoveFromService Method AddressSpace definition 65
Table 47 – PlaceInService result codes 65
Table 48 – PlaceInService Method AddressSpace definition 65
Table 49 –ShelvedStateMachineType definition 67
Table 50 – ShelvedStateMachineType transitions 68
Table 51 – Unshelve result codes 69
Table 52 – Unshelve Method AddressSpace definition 69
Table 53 – TimedShelve parameters 70
Table 54 – TimedShelve result codes 70
Table 55 – TimedShelve Method AddressSpace definition 70
Table 56 – OneShotShelve result codes 71
Table 57 – OneShotShelve Method AddressSpace definition 71
Table 58 – LimitAlarmType definition 72
Table 59 – ExclusiveLimitStateMachineType definition 74
Table 60 – ExclusiveLimitStateMachineType transitions 74
Table 61 – ExclusiveLimitAlarmType definition 76
Table 62 – NonExclusiveLimitAlarmType definition 78
Table 63 – NonExclusiveLevelAlarmType definition 79
Table 64 – ExclusiveLevelAlarmType definition 79
Table 65 – NonExclusiveDeviationAlarmType definition 80
Table 66 – ExclusiveDeviationAlarmType definition 80

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Table 67 – NonExclusiveRateOfChangeAlarmType definition 81
Table 68 – ExclusiveRateOfChangeAlarmType definition 81
Table 69 – DiscreteAlarmType definition 82
Table 70 – OffNormalAlarmType Definition 83
Table 71 – SystemOffNormalAlarmType definition 83
Table 72 – TripAlarmType definition 84
Table 73 – InstrumentDiagnosticAlarmType definition 84
Table 74 – SystemDiagnosticAlarmType definition 84
Table 75 – CertificateExpirationAlarmType definition 85
Table 76 – DiscrepancyAlarmType definition 85
Table 77 – BaseConditionClassType definition 86
Table 78 – ProcessConditionClassType definition 87
Table 79 – MaintenanceConditionClassType definition 87
Table 80 – SystemConditionClassType definition 88
Table 81 – SafetyConditionClassType definition 88
Table 82 – HighlyManagedAlarmConditionClassType definition 88
Table 83 – TrainingConditionClassType definition 89
Table 84 – StatisticalConditionClassType definition 89
Table 85 – TestingConditionSubClassType definition 89
Table 86 – AuditConditionEventType definition 91
Table 87 – AuditConditionEnableEventType definition 91
Table 88 – AuditConditionCommentEventType definition 91
Table 89 – AuditConditionRespondEventType definition 92
Table 90 – AuditConditionAcknowledgeEventType definition 92
Table 91 – AuditConditionConfirmEventType definition 93
Table 92 – AuditConditionShelvingEventType definition 93
Table 93 – AuditConditionSuppressionEventType definition 93
Table 94 – AuditConditionSilenceEventType definition 94
Table 95 – AuditConditionResetEventType definition 94
Table 96 – AuditConditionOutOfServiceEventType definition 94
Table 97 – RefreshStartEventType definition 95
Table 98 – RefreshEndEventType definition 96
Table 99 – RefreshRequiredEventType definition 96
Table 100 – HasCondition *ReferenceType* 97
Table 101 – Alarm & Condition result codes 97
Table 102 – HasEffectDisable ReferenceType 102
Table 103 – HasEffectEnable ReferenceType 103
Table 104 – HasEffectSuppress ReferenceType 103
Table 105 – HasEffectUnsuppress ReferenceType 104

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Table 106 – AlarmMetricsType Definition 105
Table 107 – AlarmRateVariableType definition 106
Table 108 – Suppress result codes 106
Table 109 – Reset Method AddressSpace definition 106
Table A.1 – Recommended state names for LocaleId "en" 107
Table A.2 – Recommended display names for LocaleId "en" 107
Table A.3 – Recommended state names for LocaleId "de" 108
Table A.4 – Recommended display names for LocaleId "de" 108
Table A.5 – Recommended state names for LocaleId "fr" 109
Table A.6 – Recommended display names for LocaleId "fr" 109
Table A.7 – Recommended dialog response options 110
Table B.1 – Example of a Condition that only keeps the latest state 112
Table B.2 – Example of a *Condition* that maintains previous states via branches 114
Table C.1 – EEMUA Terms 117
Table D.1 – Mapping from standard Event categories to OPC UA Event types 120
Table D.2 – Mapping from ONEVENTSTRUCT fields to UA BaseEventType Variables 122
Table D.3 – Mapping from ONEVENTSTRUCT fields to UA AuditEventType Variables 122
Table D.4 – Mapping from ONEVENTSTRUCT fields to UA AlarmType Variables 123
Table D.5 – Event category attribute mapping table 127
Table E.1 – IEC 62682 Mapping 134
Table F.1 – SystemStateStateMachineType definition 144
Table F.2 – SystemStateStateMachineType transitions 145

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

1 Scope

This part of IEC 62541 specifies the representation of *Alarms* and *Conditions* in the OPC Unified Architecture. Included is the *Information Model* representation of *Alarms* and *Conditions* in the OPC UA address space. Other aspects of alarm systems such as alarm philosophy, life cycle, alarm response times, alarm types and many other details are captured in documents such as IEC 62682 and ISA 18.2. The *Alarms* and *Conditions Information Model* in this specification is designed in accordance with IEC 62682 and ISA 18.2.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC TR 62541-1, *OPC unified architecture – Part 1: Overview and concepts*

IEC 62541-3, *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 62541-6, *OPC unified architecture – Part 6: Mappings*

IEC 62541-7, *OPC unified architecture – Part 7: Profiles*

IEC 62541-8, *OPC unified architecture – Part 8: Data Access*

IEC 62541-11, *OPC unified architecture – Part 11: Historical Access*

IEC 62682, *Management of alarms systems for the process industries*

EEMUA: 2nd Edition EEMUA 191 – Alarm System – A guide to design, management and procurement (Appendixes 6, 7, 8, 9), available at https://www.eemua.org/Products/Publications/Print/EEMUA-Publication-191.aspx

3 Terms, definitions, abbreviated terms and data types used

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC TR 62541-1, IEC 62541-3, IEC 62541-4, and IEC 62541-5 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1.1
Acknowledge
Operator action that indicates recognition of an *Alarm*

Note 1 to entry: This definition is copied from EEMUA. The term "Accept" is another common term used to describe *Acknowledge*. They can be used interchangeably. This document uses *Acknowledge*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

3.1.2
Active
state for an Alarm that indicates that the situation the *Alarm* is representing currently exists

Note 1 to entry: Other common terms defined by EEMUA are "Standing" for an *Active Alarm* and "Cleared" when the *Condition* has returned to normal and is no longer *Active*.

3.1.3
AdaptiveAlarm
Alarm for which the set point or limits are changed by an algorithm

Note 1 to entry: *AdaptiveAlarms* are alarms that are adjusted automatically by algorithms. These algorithms can detect conditions in a plant and change setpoints or limits to keep alarms from occurring. These changes occur, in many cases, without *Operator* interactions.

3.1.4
AlarmFlood
condition during which the alarm rate is greater than the *Operator* can effectively manage

Note 1 to entry: OPC UA does not define the conditions that would be considered alarm flooding, these conditions are defined in other specifications such as IEC 62682 or ISA 18.2.

3.1.5
AlarmSuppressionGroup
group of *Alarms* that is used to suppress other *Alarms*

Note 1 to entry: An *AlarmSuppressionGroup* is an instance of an *AlarmGroupType* that is used to suppress other *Alarms*. If any *Alarm* in the group is active, then the *AlarmSuppressionGroup* is active. If all *Alarms* in the *AlarmSuppressionGroup* are inactive then the *AlarmSuppressionGroup* is inactive

Note 2 to entry: The *Alarm* to be affected references *AlarmSuppressionGroups* with a *HasAlarmSuppressionGroup ReferenceType*.

3.1.6
ConditionClass
Condition grouping that indicates in which domain or for what purpose a certain *Condition* is used

Note 1 to entry: Some top-level *ConditionClasses* are defined in this specification. Vendors or organisations can derive more concrete classes or define different top-level classes.

3.1.7
ConditionBranch
specific state of a *Condition*

Note 1 to entry: The *Server* can maintain *ConditionBranches* for the current state as well as for previous states.

3.1.8
ConditionSource
element which a specific *Condition* is based upon or related to

Note 1 to entry: Typically, this will be a *Variable* representing a process tag (e.g. FIC101) or an *Object* representing a device or subsystem.

Note 2 to entry: In *Events* generated for *Conditions*, the *SourceNode Property* (inherited from the *BaseEventType*) will contain the *NodeId* of the *ConditionSource*.

3.1.9
confirm
Operator action informing the *Server* that a corrective action has been taken to address the cause of the *Alarm*

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

3.1.10
disable
action configurating a system such that the *Alarm* will not be generated even though the base *Alarm Condition* is present

Note 1 to entry: This definition is copied from EEMUA and is further defined in EEMUA.

Note 2 to entry: In IEC 62682, "disable" is referenced as "Out of Service".

3.1.11
LatchingAlarm
alarm that remains in alarm state after the process condition has returned to normal and requires an *Operator* reset before the alarm returns to normal

Note 1 to entry: Latching alarms are typically discrepancy alarms, where an action does not occur within a specific time. Once the action occurs the alarm stays active until it is reset.

3.1.12
Operator
special user who is assigned to monitor and control a portion of a process

Note 1 to entry: "A Member of the operations team who is assigned to monitor and control a portion of the process and is working at the control system's Console" as defined in EEMUA. In this document, an Operator is a special user. All descriptions that apply to general users also apply to Operators.

3.1.13
Refresh
act of providing an update to an *Event Subscription* that provides all *Alarms* which are considered to be *Retained*

Note 1 to entry: This concept is further defined in EEMUA.

3.1.14
Retain
Alarm in a state that is interesting for a *Client* wishing to synchronize its state of *Conditions* with the *Server*'s state

3.1.15
Shelving
facility where the *Operator* is able to temporarily prevent an *Alarm* from being displayed to the *Operator* when it is causing the *Operator* a nuisance

Note 1 to entry: "A Shelved *Alarm* will be removed from the list and will not re-annunciate until un-shelved" as defined in EEMUA.

3.1.16
Suppress
act of determining whether an *Alarm* should not occur

Note 1 to entry: "An Alarm is suppressed when logical criteria are applied to determine that the Alarm should not occur, even though the base Alarm Condition (e.g. Alarm setting exceeded) is present" as defined in EEMUA. In IEC62682 Suppressed Alarms are also described as being "Suppressed by Design", in that the system is designed with logic to Suppress an Alarm when certain criteria exist. For example, if a process unit is taken offline then low-level alarms are Suppressed for all equipment in the off-line unit.

3.2 Abbreviated terms

A&E	Alarm & Event (as used for OPC COM)
COM	(Microsoft Windows) Component Object Model
DA	data access
UA	Unified Architecture

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

3.3 Data types used

Table 1 and Table 2 describe the data types that are used throughout this document. These types are separated into two tables. Base data types defined in IEC 62541-3 are given in Table 1. The base types and data types defined in IEC 62541-4 are given in Table 2.

Table 1 – Parameter types defined in IEC 62541-3

Parameter Type
Argument
BaseDataType
NodeId
LocalizedText
Boolean
ByteString
Double
Duration
String
UInt16
Int32
UtcTime

Table 2 – Parameter types defined in IEC 62541-4

Parameter Type
IntegerId
StatusCode

4 Concepts

4.1 General

This document defines an *Information Model* for *Conditions*, Dialog *Conditions*, and *Alarms* including acknowledgement capabilities. It is built upon and extends base Event handling which is defined in IEC 62541-3, IEC 62541-4 and IEC 62541-5. This *Information Model* can also be extended to support the additional needs of specific domains. The details of which aspects of the Information Model are supported are defined via Profiles (see IEC 62541-7 for Profile definitions). Some systems may expose historical Events and Conditions via the standard Historical Access framework (see IEC 62541-11 for Historical Event definitions).

4.2 Conditions

Conditions are used to represent the state of a system or one of its components. Some common examples are:

- a temperature exceeding a configured limit;
- a device needing maintenance;
- a batch process that requires a user to confirm some step in the process before proceeding.

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

Each *Condition* instance is of a specific *ConditionType*. The *ConditionType* and derived types are subtypes of the *BaseEventType* (see IEC 62541-3 and IEC 62541-5). This part defines types that are common across many industries. It is expected that vendors or other standardisation groups will define additional *ConditionTypes* deriving from the common base types defined in this part. The *ConditionTypes* supported by a *Server* are exposed in the *AddressSpace* of the *Server*.

Condition instances are specific implementations of a *ConditionType*. It is up to the *Server* whether such instances are also exposed in the *Server's AddressSpace*. Subclause 4.10 provides additional background about *Condition* instances. *Condition* instances shall have a unique identifier to differentiate them from other instances. This is independent of whether they are exposed in the *AddressSpace*.

As mentioned above, *Conditions* represent the state of a system or one of its components. In certain cases, however, previous states that still need attention shall also be maintained. *ConditionBranches* are introduced to deal with this requirement and distinguish current state and previous states. Each *ConditionBranch* has a BranchId that differentiates it from other branches of the same *Condition* instance. The *ConditionBranch* which represents the current state of the *Condition* (the trunk) has a NULL BranchId. *Server*s can generate separate *Event Notifications* for each branch. When the state represented by a *ConditionBranch* does not need further attention, a final *Event Notification* for this branch will have the *Retain Property* set to False. Subclause 4.4 provides more information and use cases. Maintaining previous states and therefore the support of multiple branches is optional for *Servers*.

Conceptually, the lifetime of the *Condition* instance is independent of its state. However, *Servers* may provide access to *Condition* instances only while *ConditionBranches* exist.

The base *Condition* state model is illustrated in Figure 1. It is extended by the various *Condition* subtypes defined in this document and may be further extended by vendors or other standardisation groups. The primary states of a *Condition* are disabled and enabled. The *Disabled* state is intended to allow *Conditions* to be turned off at the *Server* or below the *Server* (in a device or some underlying system). The *Enabled* state is normally extended with the addition of substates.

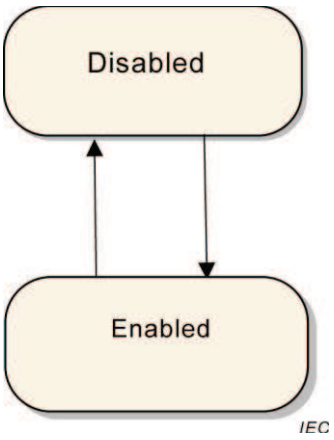

Figure 1 – Base Condition state model

A transition into the *Disable*d state results in a *Condition Event*, however no subsequent *Event Notifications* are generated until the *Condition* returns to the Enabled state.

When a *Condition* enters the Enabled state, that transition and all subsequent transitions result in *Condition Events* being generated by the *Server*.

If *Auditing* is supported by a *Server*, the following *Auditing* related action shall be performed. The *Server* will generate *AuditEvents* for *Enable* and *Disable* operations (either through a *Method* call or some *Server* / vendor – specific means), rather than generating an *AuditEvent Notification* for each *Condition* instance being enabled or disabled. For more information, see the definition of *AuditConditionEnableEventType* in 5.10.2. *AuditEvents* are also generated for any other *Operator* action that results in changes to the *Conditions*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

4.3 Acknowledgeable Conditions

AcknowledgeableConditions are subtypes of the base *ConditionType*. *AcknowledgeableConditions* expose states to indicate whether a *Condition* has to be acknowledged or confirmed.

An *AckedState* and a *ConfirmedState* extend the *EnabledState* defined by the *Condition*. The state model is illustrated in Figure 2. The enabled state is extended by adding the *AckedState* and (optionally) the *ConfirmedState*.

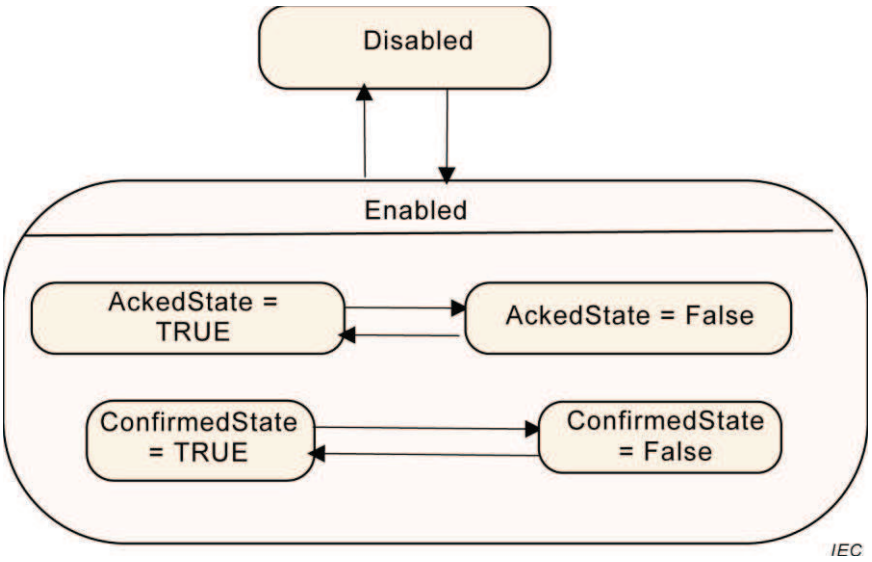

Figure 2 – AcknowledgeableConditions state model

Acknowledgment of the transition may come from the *Client* or may be due to some logic internal to the *Server*. For example, acknowledgment of a related *Condition* may result in this *Condition* becoming acknowledged, or the *Condition* may be set up to automatically *Acknowledge* itself when the acknowledgeable situation disappears.

Two *Acknowledge* state models are supported by this document. Either of these state models can be extended to support more complex acknowledgement situations.

The basic *Acknowledge* state model is illustrated in Figure 3. This model defines an AckedState. The specific state changes that result in a change to the state depend on a Server's implementation. For example, in typical *Alarm* models the change is limited to a transition to the *Active* state or transitions within the *Active* state. More complex models however can also allow for changes to the AckedState when the *Condition* transitions to an inactive state.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

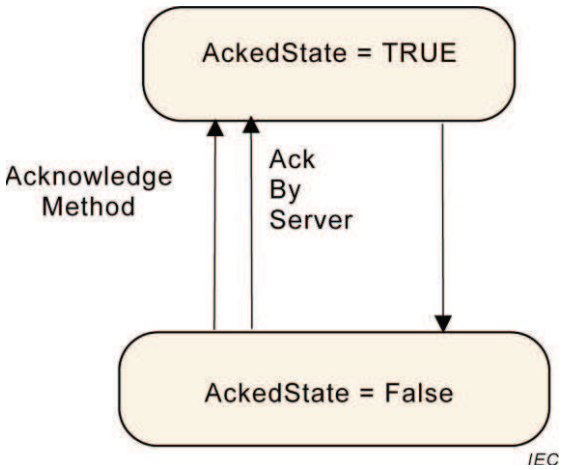

Figure 3 – Acknowledge state model

A more complex state model which adds a confirmation to the basic *Acknowledge* is illustrated in Figure 4. The *Confirmed Acknowledge* model is typically used to differentiate between acknowledging the presence of a *Condition* and having done something to address the *Condition*. For example, an *Operator* receiving a motor high temperature *Notification* calls the *Acknowledge Method* to inform the *Server* that the high temperature has been observed. The *Operator* then takes some action such as lowering the load on the motor in order to reduce the temperature. The *Operator* then calls the *Confirm Method* to inform the *Server* that a corrective action has been taken.

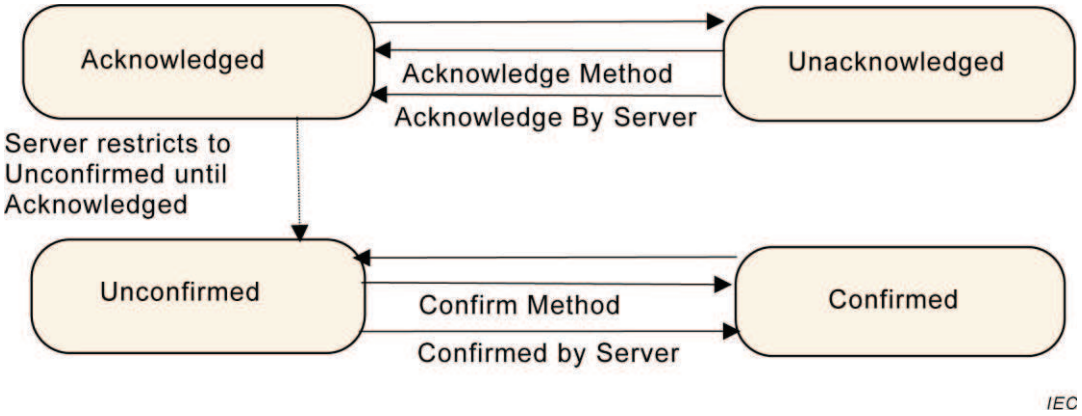

Figure 4 – Confirmed Acknowledge state model

4.4 Previous states of Conditions

Some systems require that previous states of a *Condition* are preserved for some time. A common use case is the acknowledgement process. In certain environments, it is required to acknowledge both the transition into *Active* state and the transition into an inactive state. Systems with strict safety rules sometimes require that every transition into *Active* state has to be acknowledged. In situations where state changes occur in short succession there can be multiple unacknowledged states and the *Server* has to maintain *ConditionBranches* for all previous unacknowledged states. These branches will be deleted after they have been acknowledged or if they reached their final state.

Multiple ConditionBranches can also be used for other use cases where snapshots of previous states of a *Condition* require additional actions.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

4.5 Condition state synchronization

When a *Client* subscribes for *Events*, the *Notification* of transitions will begin at the time of the *Subscription*. The currently existing state will not be reported. This means for example that *Clients* are not informed of currently *Active Alarms* until a new state change occurs.

Clients can obtain the current state of all *Condition* instances that are in an interesting state, by requesting a *Refresh* for a *Subscription*. It should be noted that *Refresh* is not a general replay capability since the *Server* is not required to maintain an *Event* history.

Clients request a *Refresh* by calling the ConditionRefresh *Method*. The *Server* will respond with a *RefreshStartEventType Event*. This *Event* is followed by the *Retained Conditions*. The *Server* may also send new *Event Notifications* interspersed with the *Refresh* related *Event Notifications*. After the *Server* is done with the *Refresh*, a *RefreshEndEvent* is issued marking the completion of the Refresh. *Clients* shall check for multiple *Event Notifications* for a *ConditionBranch* to avoid overwriting a new state delivered together with an older state from the *Refresh* process. If a *ConditionBranch* exists, then the current *Condition* shall be reported. This is True even if the only interesting item regarding the *Condition* is that *ConditionBranches* exist. This allows a *Client* to accurately represent the current *Condition* state.

A *Client* that wishes to display the current status of *Alarms* and *Conditions* (known as a "current *Alarm* display") would use the following logic to process *Refresh Event Notifications*. The *Client* flags all *Retained Conditions* as suspect on reception of the *Event* of the *RefreshStartEventType*. The *Client* adds any new *Events* that are received during the Refresh without flagging them as suspect. The *Client* also removes the suspect flag from any *Retained Conditions* that are returned as part of the Refresh. When the *Client* receives a *RefreshEndEvent*, the *Client* removes any remaining suspect *Events*, since they no longer apply.

The following items should be noted with regard to ConditionRefresh:

- As described in 4.4 some systems require that previous states of a *Condition* are preserved for some time. Some *Servers* – in particular if they require acknowledgement of previous states – will maintain separate *ConditionBranches* for prior states that still need attention.

  ConditionRefresh shall issue *Event Notifications* for all interesting states (current and previous) of a *Condition* instance and *Clients* can therefore receive more than one *Event* for a *Condition* instance with different BranchIds.
- Under some circumstances a *Server* may not be capable of ensuring the *Client* is fully in sync with the current state of *Condition* instances. For example, if the underlying system represented by the *Server* is reset or communications are lost for some period of time the *Server* may need to resynchronize itself with the underlying system. In these cases, the *Server* shall send an *Event* of the *RefreshRequiredEventType* to advise the *Client* that a *Refresh* may be necessary. A *Client* receiving this special *Event* should initiate a *ConditionRefresh* as noted in this subclause.
- To ensure a *Client* is always informed, the three special *EventTypes* (*RefreshEndEventType*, *RefreshStartEventType* and *RefreshRequiredEventType*) ignore the *Event* content filtering associated with a *Subscription* and will always be delivered to the *Client*.
- ConditionRefresh applies to a *Subscription*. If multiple Event Notifiers are included in the same *Subscription*, all *Event Notifiers* are refreshed.

4.6 Severity, quality, and comment

Comment, severity and quality are important elements of *Conditions* and any change to them will cause *Event Notification*s.

The Severity of a *Condition* is inherited from the base *Event* model defined in IEC 62541-5. It indicates the urgency of the *Condition* and is also commonly called "priority", especially in relation to *Alarms* in the *ProcessConditionClassType*.

A Comment is a user generated string that is to be associated with a certain state of a *Condition*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Quality refers to the quality of the data value(s) upon which this *Condition* is based. Since a *Condition* is usually based on one or more *Variables*, the *Condition* inherits the quality of these *Variables*. E.g., if the process value is "Uncertain", the "Level Alarm" *Condition* is also questionable. If more than one variable is represented by a given condition or if the condition is from an underlining system and no direct mapping to a variable is available, it is up to the application to determine what quality is displayed as part of the condition.

4.7 Dialogs

Dialogs are *ConditionTypes* used by a *Server* to request user input. They are typically used when a *Server* has entered some state that requires intervention by a *Client*. For example a *Server* monitoring a paper machine indicates that a roll of paper has been wound and is ready for inspection. The *Server* would activate a Dialog *Condition* indicating to the user that an inspection is required. Once the inspection has taken place, the user responds by informing the *Server* of an accepted or unaccepted inspection allowing the process to continue.

4.8 Alarms

Alarms are specializations of *AcknowledgeableConditions* that add the concepts of an *Active* state and other states like *Shelving* state and *Suppressed* state to a *Condition*. The state model is illustrated in Figure 5. The complete model with all states is defined in 5.8.

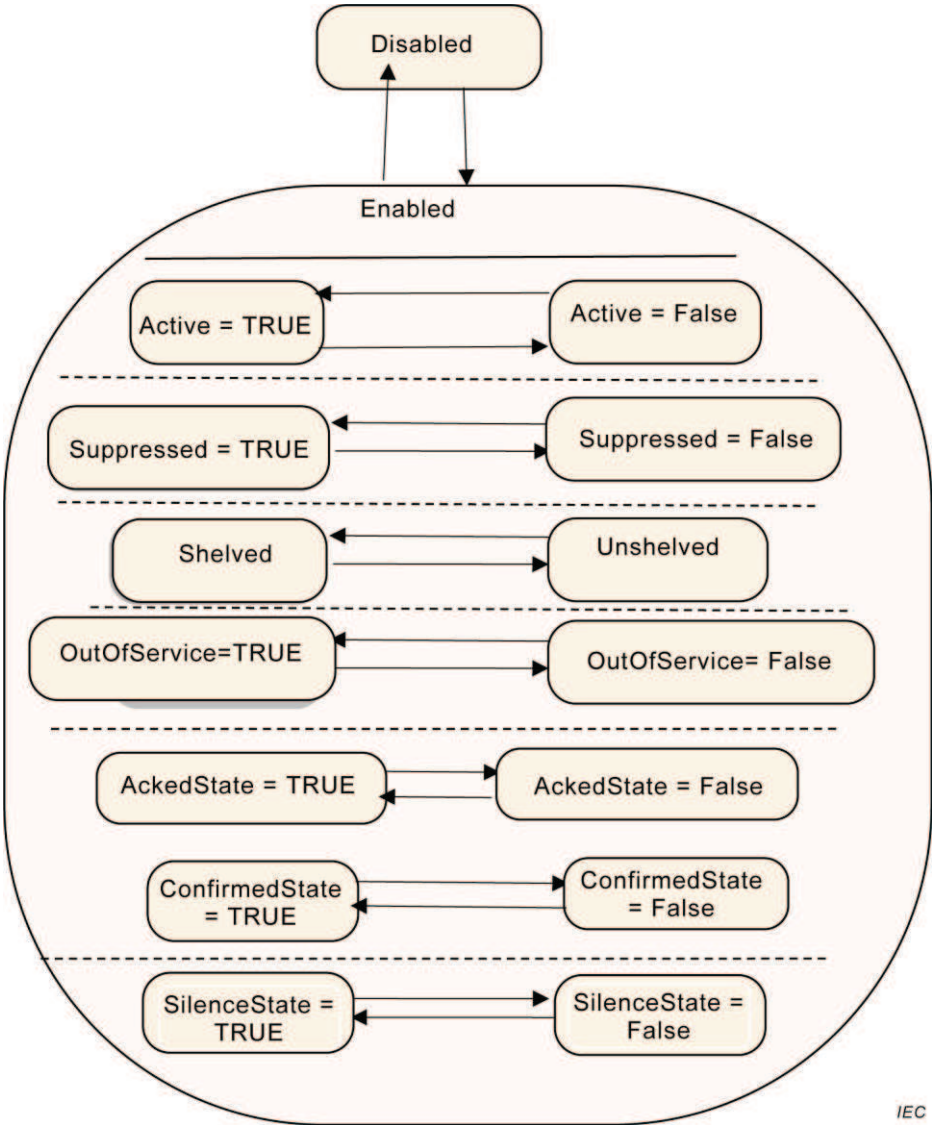

Figure 5 – Alarm state machine model

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

An *Alarm* in the *Active* state indicates that the situation the *Condition* is representing currently exists. When an *Alarm* is an inactive state it is representing a situation that has returned to a normal state.

Some *Alarm* subtypes introduce substates of the *Active* state. For example, an *Alarm* representing a temperature may provide a high-level state as well as a critically high state (see following Clause).

The *Shelving* state can be set by an *Operator* via OPC UA *Methods*. The *Suppressed* state is set internally by the *Server* due to system specific reasons. *Alarm* systems typically implement the suppress, out of service and shelve features to help keep *Operators* from being overwhelmed during *Alarm* "storms" by limiting the number of *Alarms* an *Operator* sees on a current *Alarm* display. This is accomplished by setting the *SuppressedOrShelved* flag on second order dependent *Alarms* and/or *Alarms* of less severity, leading the *Operator* to concentrate on the most critical issues.

The shelved, out of service and suppressed states differ from the *Disabled* state in that *Alarms* are still fully functional and can be included in *Subscription Notifications* to a *Client*.

Alarms follow a typical timeline, which is illustrated in Figure 6. They have a number of delay times associated with them and a number of states that they might occupy. The goal of an alarming system is to inform *Operator*s about conditions in a timely manner and allow the *Operator* to take some action before some consequences occur. The consequences can be economic (product is not usable and shall be discarded), can be physical (tank overflows), can be safety related (fire or explosion could occur) or any of a number of other possibilities. Typically, if no action is taken related to an alarm for some period of time, the process will cross some threshold at which point consequences will start to occur. The OPC UA *Alarm* model describes these states, delays and actions.

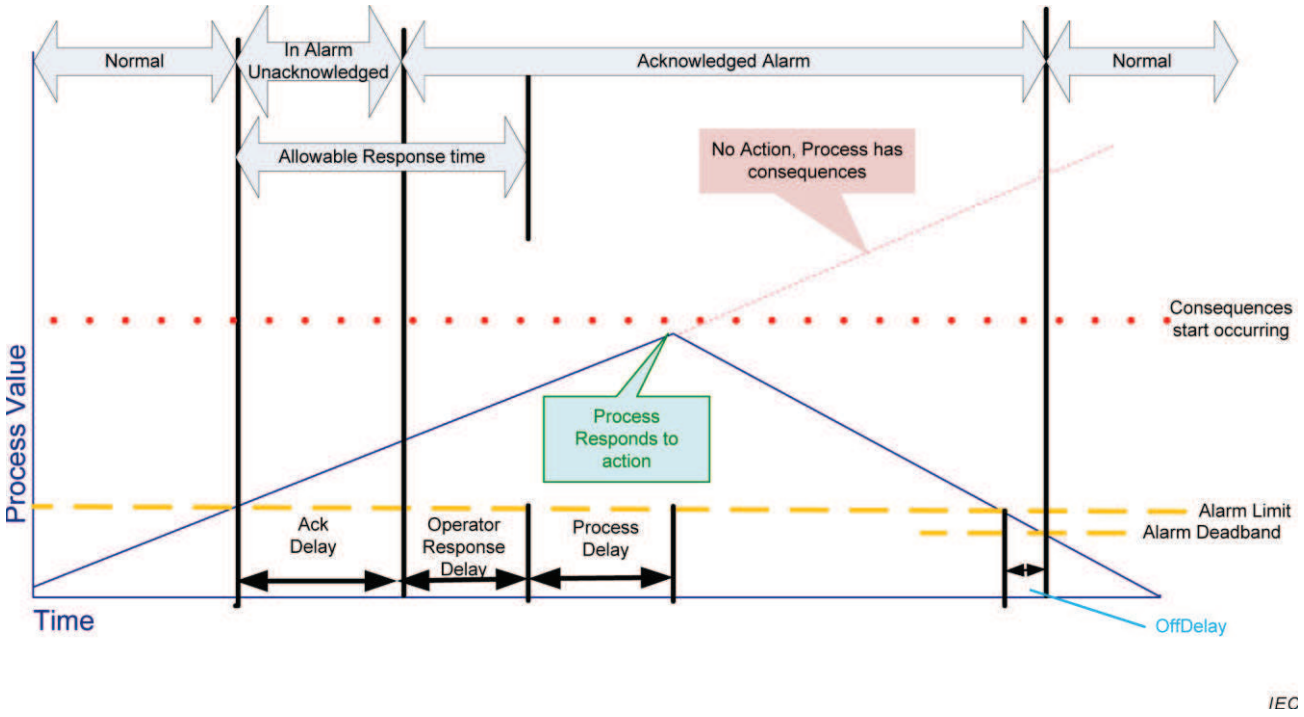

Figure 6 – Typical Alarm Timeline example

4.9 Multiple active states

In some cases, it is desirable to further define the *Active* state of an *Alarm* by providing a substate machine for the *Active* State. For example, a multi-state level *Alarm* when in the *Active* state may be in one of the following substates: LowLow, Low, High or HighHigh. The state model is illustrated in Figure 7.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

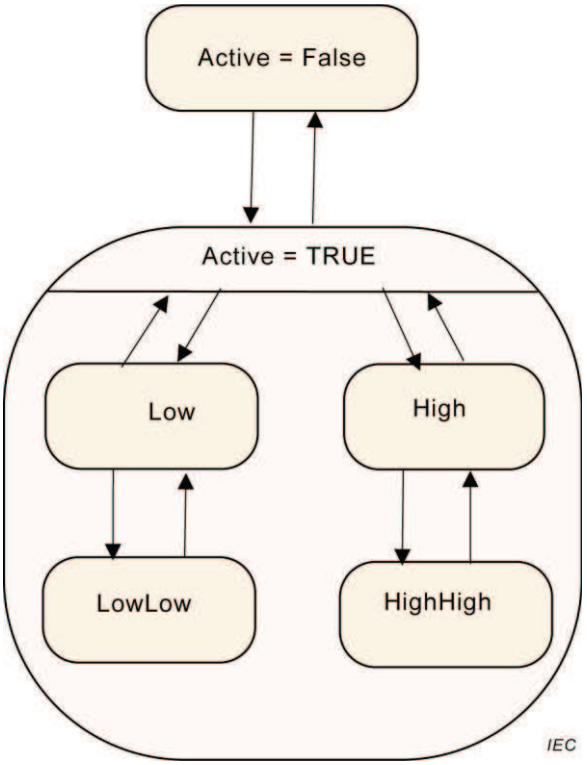

Figure 7 – Multiple active states example

With the multi-state *Alarm* model, state transitions among the substates of *Active* are allowed without causing a transition out of the *Active* state.

To accommodate different use cases both a (mutually) exclusive and a non-exclusive model are supported.

Exclusive means that the *Alarm* can only be in one substate at a time. If for example a temperature exceeds the HighHigh limit the associated exclusive level *Alarm* will be in the HighHigh substate and not in the High substate.

Some *Alarm* systems, however, allow multiple substates to exist in parallel. This is called non-exclusive. In the previous example where the temperature exceeds the HighHigh limit a non-exclusive level *Alarm* will be both in the High and the HighHigh substate.

4.10 Condition instances in the AddressSpace

Because *Conditions* always have a state (*Enabled* or *Disabled*) and possibly many substates it makes sense to have instances of *Conditions* present in the *AddressSpace*. If the *Server* exposes *Condition* instances they usually will appear in the *AddressSpace* as components of the *Objects* that "own" them. For example, a temperature transmitter that has a built-in high temperature *Alarm* would appear in the *AddressSpace* as an instance of some temperature transmitter *Object* with a *HasComponent Reference* to an instance of a *LimitAlarmType*.

The availability of instances allows Data Access *Clients* to monitor the current *Condition* state by subscribing to the *Attribute* values of *Variable Nodes*. The values of the nodes may not always correspond with the value that appear in *Events*, they may be more recent than what was in the *Event*.

While exposing *Condition* instances in the *AddressSpace* is not always possible, doing so allows for direct interaction (read, write and *Method* invocation) with a specific *Condition* instance. For example, if a *Condition* instance is not exposed, there is no way to invoke the *Enable* or *Disable Method* for the specific *Condition* instance.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

4.11 Alarm and Condition auditing

The IEC 62541 series includes provisions for auditing. Auditing is an important security and tracking concept. Audit records provide the "Who", "When" and "What" information regarding user interactions with a system. These audit records are especially important when *Alarm* management is considered. *Alarms* are the typical instrument for providing information to a user that something needs the user's attention. A record of how the user reacts to this information is required in many cases. Audit records are generated for all *Method* calls that affect the state of the system, for example, an *Acknowledge Method* call would generate an *AuditConditionAcknowledgeEventType Event*.

The standard *AuditEventTypes* defined in IEC 62541-5 already include the fields required for *Condition* related audit records. To allow for filtering and grouping, this document defines a number of subtypes of the *AuditEventTypes* but without adding new fields to them.

This document describes the *AuditEventType* that each *Method* is required to generate. For example, the *Disable Method* has an *AlwaysGeneratesEvent Reference* to an *AuditConditionEnableEventType*. An *Event* of this type shall be generated for every invocation of the *Method*. The audit *Event* describes the user interaction with the system, in some cases this interaction may affect more than one *Condition* or be related to more than one state.

5 Model

5.1 General

The *Alarm* and *Condition* model extends the OPC UA base *Event* model by defining various *Event Types* based on the *BaseEventType*. All of the *Event Types* defined in this document can be further extended to form domain or *Server* specific *Alarm* and *Condition Types*.

Instances of *Alarm* and *Condition Types* may be optionally exposed in the *AddressSpace* in order to allow direct access to the state of an *Alarm* or *Condition*.

Subclauses 5.5 to 5.8 define the OPC UA *Alarm* and *Condition Types*. Figure 8 informally describes the hierarchy of these *Types*. Subtypes of the *LimitAlarmType* and the *DiscreteAlarmType* are not shown. The full *AlarmConditionType* hierarchy can be found in Figure 8.

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

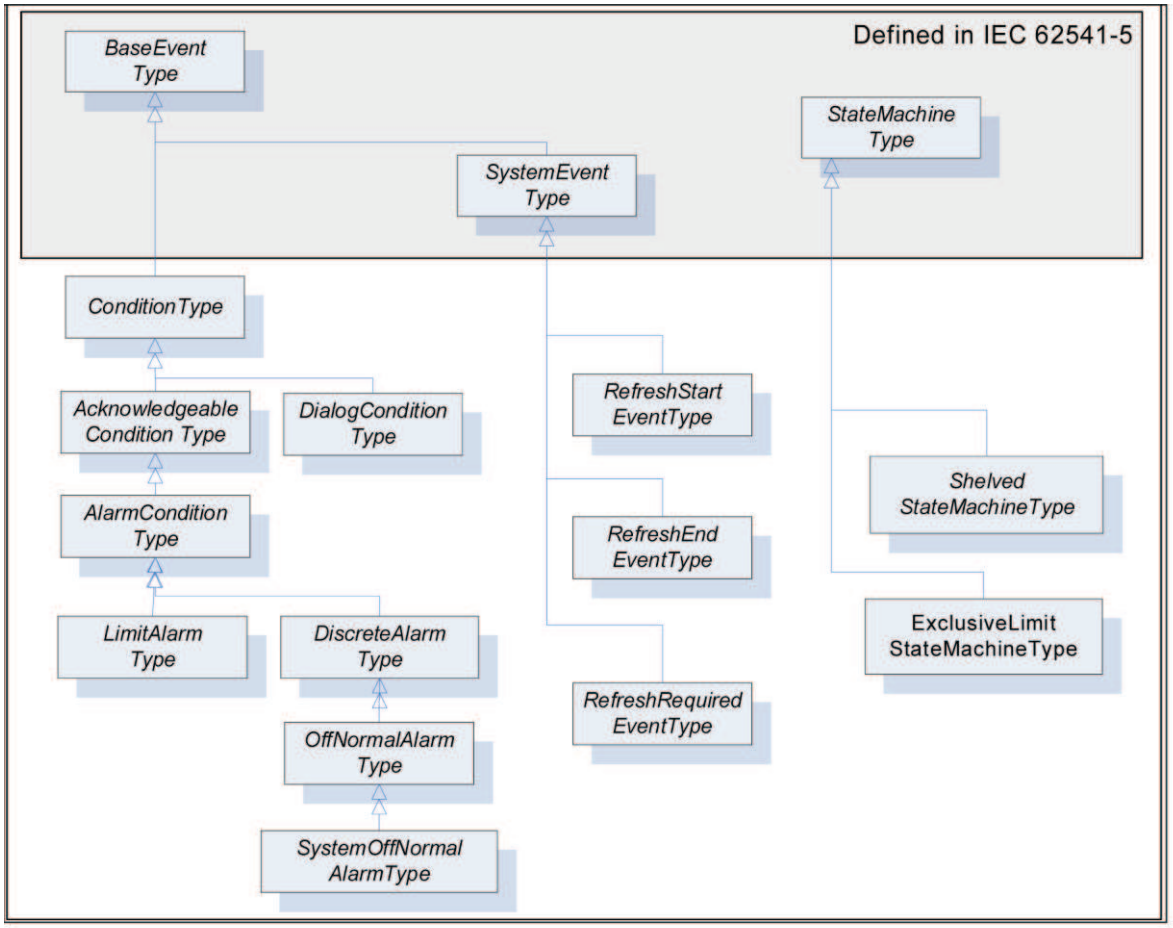

Figure 8 – ConditionType hierarchy

Annex C specifies how the model described in this document maps to EEMUA.

Annex D specifies a recommended mapping between OPC Classic Alarm & Events (A&E) servers and the model described in this document.

5.2 Two-state state machines

Most states defined in this document are simple – i.e. they are either True or False. The *TwoStateVariableType* is introduced specifically for this use case. More complex states are modelled by using a *StateMachineType* defined in IEC 62541-5.

The *TwoStateVariableType* is derived from the *StateVariableType* defined in IEC 62541-5 and formally defined in Table 3.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Table 3 – TwoStateVariableType definition

Attribute	Value				
BrowseName	TwoStateVariableType				
DataType	LocalizedText				
ValueRank	-1 (-1 = Scalar)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *StateVariableType* defined in IEC 62541-5. Note that a *Reference* to this subtype is not shown in the definition of the *StateVariableType*					
HasProperty	Variable	Id	Boolean	PropertyType	Mandatory
HasProperty	Variable	TransitionTime	UtcTime	PropertyType	Optional
HasProperty	Variable	EffectiveTransitionTime	UtcTime	PropertyType	Optional
HasProperty	Variable	TrueState	LocalizedText	PropertyType	Optional
HasProperty	Variable	FalseState	LocalizedText	PropertyType	Optional
HasTrueSubState	StateMachine or TwoStateVariableType	<StateIdentifier>	Defined in Clause 5.4.2		Optional
HasFalseSubState	StateMachine or TwoStateVariableType	<StateIdentifier>	Defined in Clause 5.4.3		Optional

The *Value Attribute* of an instance of *TwoStateVariableType* contains the current state as a human readable name. The *EnabledState* for example, might contain the name "Enabled" when True and "Disabled" when False.

Id is inherited from the *StateVariableType* and overridden to reflect the required *DataType* (Boolean). The value shall be the current state, i.e. either True or False.

TransitionTime specifies the time when the current state was entered.

EffectiveTransitionTime specifies the time when the current state or one of its substates was entered. If, for example, a LevelAlarm is active and – while active – switches several times between High and HighHigh, then the *TransitionTime* stays at the point in time where the *Alarm* became active whereas the *EffectiveTransitionTime* changes with each shift of a substate.

The optional *Property EffectiveDisplayName* from the *StateVariableType* is used if a state has substates. It contains a human readable name for the current state after taking the state of any *SubStateMachines* in account. As an example, the *EffectiveDisplayName* of the *EnabledState* could contain "Active/HighHigh" to specify that the *Condition* is active and has exceeded the HighHigh limit.

Other optional *Properties* of the *StateVariableType* have no defined meaning for *TwoStateVariableType*.

TrueState and *FalseState* contain the localized string for the *TwoStateVariableType* value when its *Id Property* has the value True or False, respectively. Since the two *Properties* provide meta-data for the *Type*, *Servers* may not allow these *Properties* to be selected in the *Event* filter for a MonitoredItem. *Clients* can use the Read *Service* to get the information from the specific *ConditionType*.

A *HasTrueSubState Reference* is used to indicate that the True state has substates.

A *HasFalseSubState Reference* is used to indicate that the False state has substates.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

5.3 ConditionVariable

Various information elements of a *Condition* are not considered to be states. However, a change in their value is considered important and supposed to trigger an *Event Notification*. These information elements are called *ConditionVariable*.

ConditionVariables are represented by a *ConditionVariableType,* formally defined in Table 4.

Table 4 – ConditionVariableType definition

Attribute	Value				
BrowseName	ConditionVariableType				
DataType	BaseDataType				
ValueRank	-2 (-2 = Any)				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseDataVariableType* defined in IEC 62541-5.					
HasProperty	Variable	SourceTimestamp	UtcTime	PropertyType	Mandatory

SourceTimestamp indicates the time of the last change of the *Value* of this *ConditionVariable*. It shall be the same time that would be returned from the *Read Service* inside the *DataValue* structure for the *ConditionVariable Value Attribute*.

5.4 ReferenceTypes

5.4.1 General

This Clause defines *ReferenceTypes* that are needed beyond those already specified as part of IEC 62541-3 and IEC 62541-5. This includes extending *TwoStateVariableType* state machines with substates and the addition of *Alarm* grouping.

The *TwoStateVariableType References* will only exist when substates are available. For example, if a *TwoStateVariableType* machine is in a False State, then any substates referenced from the True state will not be available. If an Event is generated while in the False state and information from the True state substate is part of the data that is to be reported than this data would be reported as a NULL. With this approach, *TwoStateVariableTypes* can be extended with subordinate state machines in a similar fashion to the *StateMachineType* defined in IEC 62541-5.

5.4.2 HasTrueSubState ReferenceType

The HasTrueSubState ReferenceType is a concrete ReferenceType that can be used directly. It is a subtype of the NonHierarchicalReferences ReferenceType.

The semantics indicate that the substate (the target Node) is a subordinate state of the True super state. If more than one state within a Condition is a substate of the same super state (i.e. several HasTrueSubState References exist for the same super state) they are all treated as independent substates. The representation in the AddressSpace is specified in Table 5.

The SourceNode of the Reference shall be an instance of a TwoStateVariableType and the TargetNode shall be either an instance of a TwoStateVariableType or an instance of a subtype of a StateMachineType.

It is not required to provide the HasTrueSubState Reference from super state to substate, but it is required that the substate provides the inverse Reference (IsTrueSubStateOf) to its super state.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Table 5 – HasTrueSubState ReferenceType

Attributes	Value		
BrowseName	HasTrueSubState		
InverseName	IsTrueSubStateOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

5.4.3 HasFalseSubState ReferenceType

The HasFalseSubState ReferenceType is a concrete ReferenceType that can be used directly. It is a subtype of the NonHierarchicalReferences ReferenceType.

The semantics indicate that the substate (the target Node) is a subordinate state of the False super state. If more than one state within a Condition is a substate of the same super state (i.e. several HasFalseSubState References exist for the same super state) they are all treated as independent substates. The representation in the AddressSpace is specified in Table 6.

The SourceNode of the Reference shall be an instance of a TwoStateVariableType and the TargetNode shall be either an instance of a TwoStateVariableType or an instance of a subtype of a StateMachineType.

It is not required to provide the HasFalseSubState Reference from super state to substate, but it is required that the substate provides the inverse Reference (IsFalseSubStateOf) to its super state.

Table 6 – HasFalseSubState ReferenceType

Attributes	Value		
BrowseName	HasFalseSubState		
InverseName	IsFalseSubStateOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

5.4.4 HasAlarmSuppressionGroup ReferenceType

The *HasAlarmSuppressionGroup ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *HasComponent ReferenceType*.

This *ReferenceType* binds an AlarmSuppressionGroup to an *Alarm*.

The *SourceNode* of the *Reference* shall be an instance of an *AlarmConditionType* or subtype and the *TargetNode* shall be an instance of an *AlarmGroupType*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Table 7 – HasAlarmSuppressionGroup ReferenceType

Attributes	Value		
BrowseName	HasAlarmSuppressionGroup		
InverseName	IsAlarmSuppressionGroupOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

5.4.5 AlarmGroupMember ReferenceType

The *AlarmGroupMember ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *Organizes* Reference Type.

This *ReferenceType* is used to indicate the *Alarm* instances that are part of an *Alarm Group*.

The *SourceNode* of the *Reference* shall be an instance of an *AlarmGroupType* or subtype of it and the *TargetNode* shall be an instance of an *AlarmConditionType* or a subtype of it.

Table 8 – AlarmGroupMember ReferenceType

Attributes	Value		
BrowseName	AlarmGroupMember		
InverseName	MemberOfAlarmGroup		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**

5.5 Condition Model

5.5.1 General

The *Condition* model extends the *Event* model by defining the *ConditionType*. The *ConditionType* introduces the concept of states differentiating it from the base *Event* model. Unlike the *BaseEventType*, *Conditions* are not transient. The *ConditionType* is further extended into Dialog and *AcknowledgeableConditionType*, each of which has its own subtypes.

The *Condition* model is illustrated in Figure 9 and formally defined in the subsequent tables. It is worth noting that this figure, like all figures in this document, is not intended to be complete. Rather, the figures only illustrate information provided by the formal definitions.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

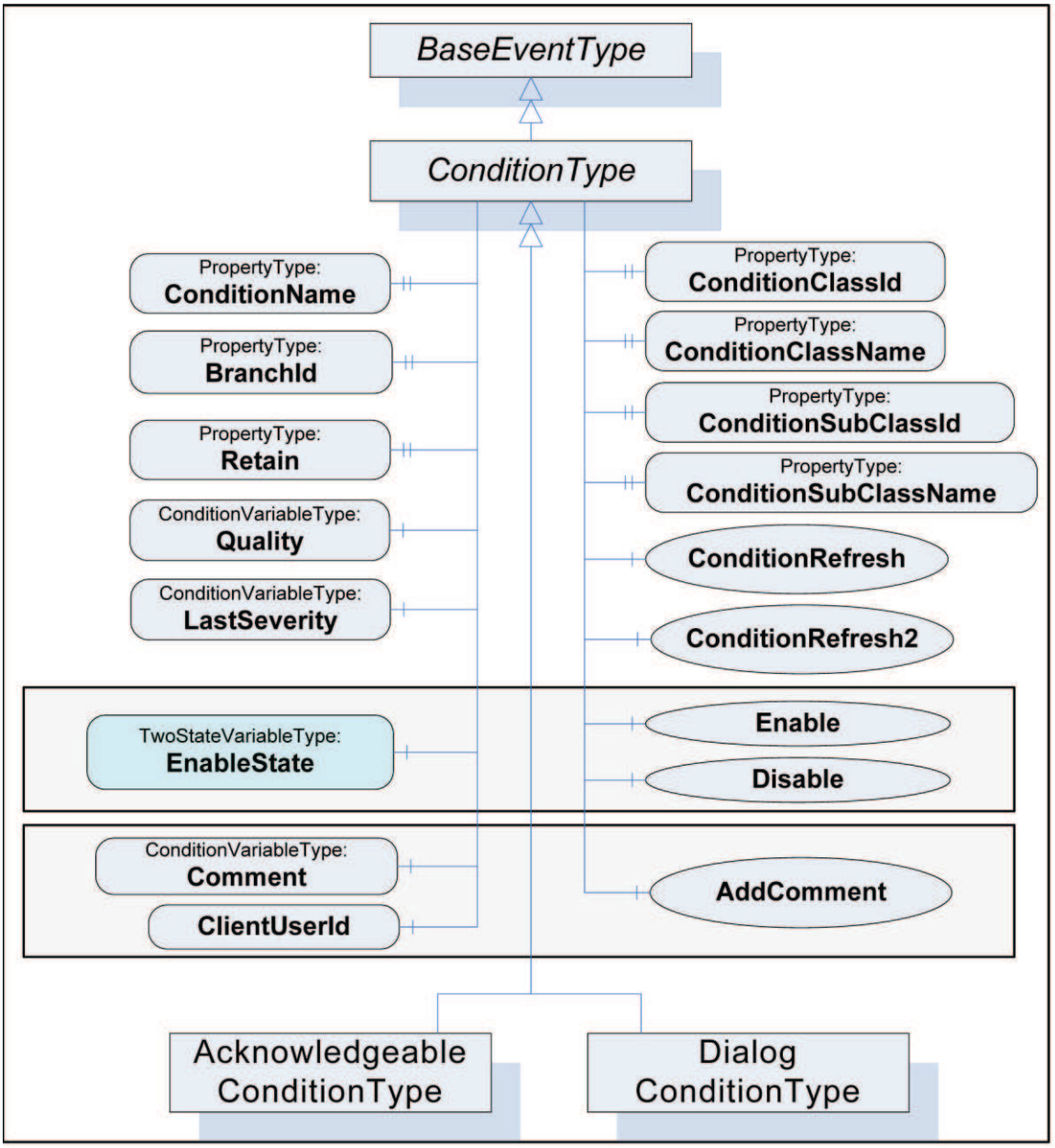

Figure 9 – Condition model

5.5.2 ConditionType

The *ConditionType* defines all general characteristics of a *Condition*. All other *ConditionTypes* derive from it. It is formally defined in Table 9. The False state of the *EnabledState* shall not be extended with a substate machine.

Table 9 – ConditionType definition

Attribute	Value
BrowseName	ConditionType
IsAbstract	True

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Attribute	Value				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
Subtype of the *BaseEventType* defined in IEC 62541-5					
HasSubtype	ObjectType	DialogConditionType	Defined in 5.6.2		
HasSubtype	ObjectType	AcknowledgeableConditionType	Defined in 5.7.2		
HasProperty	Variable	ConditionClassId	NodeId	PropertyType	Mandatory
HasProperty	Variable	ConditionClassName	LocalizedText	PropertyType	Mandatory
HasProperty	Variable	ConditionSubClassId	NodeId[]	PropertyType	Optional
HasProperty	Variable	ConditionSubClassName	LocalizedText[]	PropertyType	Optional
HasProperty	Variable	ConditionName	String	PropertyType	Mandatory
HasProperty	Variable	BranchId	NodeId	PropertyType	Mandatory
HasProperty	Variable	Retain	Boolean	PropertyType	Mandatory
HasComponent	Variable	EnabledState	LocalizedText	TwoStateVariableType	Mandatory
HasComponent	Variable	Quality	StatusCode	ConditionVariableType	Mandatory
HasComponent	Variable	LastSeverity	UInt16	ConditionVariableType	Mandatory
HasComponent	Variable	Comment	LocalizedText	ConditionVariableType	Mandatory
HasProperty	Variable	ClientUserId	String	PropertyType	Mandatory
HasComponent	Method	Disable	Defined in 5.5.4		Mandatory
HasComponent	Method	Enable	Defined in 5.5.5		Mandatory
HasComponent	Method	AddComment	Defined in 5.5.6		Mandatory
HasComponent	Method	ConditionRefresh	Defined in 5.5.7		None
HasComponent	Method	ConditionRefresh2	Defined in 5.5.8		None

The *ConditionType* inherits all *Properties* of the *BaseEventType*. Their semantic is defined in IEC 62541-5. *SourceNode Property* identifies the *ConditionSource*. See 5.12 for more details. If the *ConditionSource* is not a *Node* in the *AddressSpace,* the *NodeId* is set to NULL. The *SourceNode* Property is the *Node,* which the *Condition* is associated with, it may be the same as the *InputNode* for an *Alarm*, but it may be a separate node. For example, a motor, which is a *Variable* with a *Value* that is an RPM, may be the *ConditionSource* for *Conditions* which are related to the motor as well as a temperature sensor associated with the motor. In the former, the *InputNode* for the High RPM *Alarm* is the value of the Motor RPM, while in the later the *InputNode* of the High Alarm would be the value of the temperature sensor that is associated with the motor.

ConditionClassId specifies in which domain this *Condition* is used. It is the *NodeId* of the corresponding subtype of *BaseConditionClassType*. See 5.9 for the definition of *ConditionClass* and a set of *ConditionClasses* defined in this document. When using this *Property* for filtering, *Clients* shall specify all individual subtypes of *BaseConditionClassType NodeIds*. The *OfType* operator cannot be applied. *BaseConditionClassType* is used as class whenever a *Condition* cannot be assigned to a more concrete class.

ConditionClassName provides the display name of the subtype of *BaseConditionClassType*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

ConditionSubClassId specifies additional class[es] that apply to the *Condition.* It is the *NodeId* of the corresponding subtype of *BaseConditionClassType*. See 5.9.6 for the definition of *ConditionClass* and a set of *ConditionClasses* defined in this document. When using this *Property* for filtering, *Clients* shall specify all individual subtypes of *BaseConditionClassType NodeIds*. The OfType operator cannot be applied. The *Client* specifies a NULL in the filter, to return *Conditions* where no sub class is applied. When returning *Conditions*, if this optional field is not available in a *Condition*, a NULL shall be returned for the field.

ConditionSubClassName provides the display name[s] of the *ConditionClassType[s]* listed in the ConditionSubClassId.

ConditionName identifies the *Condition* instance that the *Event* originated from. It can be used together with the *SourceName* in a user display to distinguish between different *Condition* instances. If a *ConditionSource* has only one instance of a *ConditionType*, and the *Server* has no instance name, the *Server* shall supply the *ConditionType* browse name.

BranchId is NULL for all *Event Notifications* that relate to the current state of the *Condition* instance. If *BranchId* is not NULL, it identifies a previous state of this *Condition* instance that still needs attention by an *Operator*. If the current *ConditionBranch* is transformed into a previous *ConditionBranch* then the Server needs to assign a non-NULL *BranchId*. An initial *Event* for the branch will generated with the values of the *ConditionBranch* and the new *BranchId*. The *ConditionBranch* can be updated many times before it is no longer needed. When the *ConditionBranch* no longer requires *Operator* input the final *Event* will have *Retain* set to False. The retain bit on the current Event is True, as long as any ConditionBranches require Operator input. See 4.4 for more information about the need for creating and maintaining previous *ConditionBranches* and Clause B.1 for an example using branches. The *BranchId DataType* is *NodeId* although the *Server* is not required to have *ConditionBranches* in the *Address Space*. The use of a *NodeId* allows the *Server* to use simple numeric identifiers, strings or arrays of bytes.

Retain when True describes a *Condition* (or *ConditionBranch*) as being in a state that is interesting for a *Client* wishing to synchronize its state with the *Server*'s state. The logic to determine how this flag is set is *Server* specific. Typically, all *Active Alarms* would have the *Retain* flag set; however, it is also possible for inactive *Alarms* to have their *Retain* flag set to TRUE.

In normal processing when a *Client* receives an *Event* with the *Retain* flag set to False, the *Client* should consider this as a *ConditionBranch* that is no longer of interest, in the case of a "current *Alarm* display" the *ConditionBranch* would be removed from the display.

EnabledState indicates whether the *Condition* is enabled. *EnabledState/Id* is True if enabled, False otherwise. *EnabledState/TransitionTime* defines when the *EnabledState* last changed. Recommended state names are described in Annex A.

A *Condition*'s *EnabledState* effects the generation of *Event Notifications* and as such results in the following specific behaviour:

- When the *Condition* instance enters the *Disabled* state, the *Retain Property* of this *Condition* shall be set to False by the *Server* to indicate to the *Client* that the *Condition* instance is currently not of interest to *Client*s. This includes all *ConditionBranches* if any branches exist.
- When the *Condition* instance enters the enabled state, the *Condition* shall be evaluated and all of its *Properties* updated to reflect the current values. If this evaluation causes the *Retain Property* to transition to True for any *ConditionBranch*, then an *Event Notification* shall be generated for that *ConditionBranch*.
- The *Server* may choose to continue to test for a *Condition* instance while it is *Disable*d. However, no *Event Notifications* will be generated while the *Condition* instance is disabled.
- For any *Condition* that exists in the *AddressSpace* the *Attributes* and the following *Variables* will continue to have valid values even in the *Disable*d state; EventId, *Event* Type, Source *Node*, Source Name, Time, and EnabledState. Other *Properties* may no longer provide current valid values. All *Variables* that are no longer provided shall return a status of Bad_ConditionDisabled. The Event that reports the Disabled state should report the *Properties* as NULL or with a status of Bad_ConditionDisabled.

When enabled, changes to the following components shall cause a *ConditionType Event Notification*:

Documents
Document Sharing Hub

Full document details: DIN EN IEC 62541-9:2021-08

- Quality
- Severity (inherited from *BaseEventType*)
- Comment

This may not be the complete list. Subtypes may define additional *Variables* that trigger *Event Notifications*. In general, changes to *Variables* of the types *TwoStateVariableType* or *ConditionVariableType* trigger *Event Notifications*.

Quality reveals the status of process values or other resources that this *Condition* instance is based upon. If, for example, a process value is "Uncertain", the associated "LevelAlarm" *Condition* is also questionable. Values for the *Quality* can be any of the OPC *StatusCodes* defined in IEC 62541-8 as well as *Good*, *Uncertain* and *Bad* as defined in IEC 62541-4. These *StatusCodes* are similar to but slightly more generic than the description of data quality in the various field bus specifications. It is the responsibility of the *Server* to map internal status information to these codes. A *Server* that supports no quality information shall return *Good*. This quality can also reflect the communication status associated with the system that this value or resource is based on and from which this *Alarm* was received. For communication errors to the underlying system, especially those that result in some unavailable *Event* fields, the quality shall be Bad_NoCommunication error.

Events are only generated for *Conditions* that have their *Retain* field set to True and for the initial transition of the *Retain* field from True to False.

LastSeverity provides the previous severity of the *ConditionBranch*. Initially this *Variable* contains a zero value; it will return a value only after a severity change. The new severity is supplied via the *Severity Property,* which is inherited from the *BaseEventType*.

Comment contains the last comment provided for a certain state (*ConditionBranch*). It may have been provided by an *AddComment Method,* some other *Method* or in some other manner. The initial value of this *Variable* is NULL, unless it is provided in some other manner. If a Method provides as an option the ability to set a *Comment*, then the value of this *Variable* is reset to NULL if an optional comment is not provided.

ClientUserId is related to the *Comment* field and contains the identity of the user who inserted the most recent *Comment.* The logic to obtain the *ClientUserId* is defined in IEC 62541-5.

The *NodeId* of the *Condition* instance is used as *ConditionId*. It is not explicitly modelled as a component of the *ConditionType*. However, it can be requested with the following *SimpleAttributeOperand* (see Table 10) in the *SelectClause* of the *EventFilter*:

Table 10 – SimpleAttributeOperand

Name	Type	Description
SimpleAttributeOperand		
typeId	NodeId	*NodeId* of the *ConditionType Node*
browsePath[]	QualifiedName	empty
attributeId	IntegerId	Id of the *NodeId Attribute*

5.5.3 Condition and branch instances

Conditions are *Objects* which have a state which changes over time. Each *Condition* instance has the *ConditionId* as identifier which uniquely identifies it within the *Server*.

A *Condition* instance may be an *Object* that appears in the *Server Address Space*. If this is the case the *ConditionId* is the *NodeId* for the *Object*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

The state of a *Condition* instance at any given time is the set values for the *Variable*s that belong to the *Condition* instance. If one or more *Variable* values change the *Server* generates an *Event* with a unique *EventId*.

If a *Client* calls *Refresh* the *Server* will report the current state of a *Condition* instance by re-sending the last *Event* (i.e. the same *EventId* and *Time* is sent).

A *ConditionBranch* is a copy of the *Condition* instance state that can change independently of the current *Condition* instance state. Each *Branch* has an identifier called a *BranchId* which is unique among all active *Branches* for a *Condition* instance. *Branches* are typically not visible in the *Address Space* and this document does not define a standard way to make them visible.

5.5.4 Disable Method

The *Disable Method* is used to change a *Condition* instance to the *Disabled* state. Normally, the *NodeId* of the object instance as the *ObjectId* is passed to the Call *Service*. However, some *Servers* do not expose *Condition* instances in the *AddressSpace.* Therefore, all *Servers* shall allow *Clients* to call the *Disable Method* by specifying *ConditionId* as the *ObjectId.* The *Method* cannot be called with an *ObjectId* of the *ConditionType Node*.

Signature

```
Disable();
```

Method Result Codes in Table 11 (defined in Call *Service*).

Table 11 – Disable result codes

Result Code	Description
Bad_ConditionAlreadyDisabled	See Table 101 for the description of this result code.

Table 12 specifies the *AddressSpace* representation for the *Disable Method*.

Table 12 – Disable Method AddressSpace definition

Attribute	Value				
BrowseName	Disable				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
AlwaysGeneratesEvent	ObjectType	AuditConditionEnableEventType	Defined in 5.10.2		

5.5.5 Enable Method

The *Enable Method* is used to change a *Condition* instance to the enabled state. Normally, the *NodeId* of the object instance as the *ObjectId* is passed to the Call *Service*. However, some *Servers* do not expose *Condition* instances in the *AddressSpace.* Therefore, all *Servers* shall allow *Clients* to call the *Enable Method* by specifying *ConditionId* as the *ObjectId.* The *Method* cannot be called with an *ObjectId* of the *ConditionType Node*. If the *Condition* instance is not exposed, then it may be difficult for a *Client* to determine the *ConditionId* for a disabled *Condition.*

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Signature

```
Enable();
```

Method result codes in Table 13 (defined in Call Service).

Table 13 – Enable result codes

Result Code	Description
Bad_ConditionAlreadyEnabled	See Table 101 for the description of this result code.

Table 14 specifies the *AddressSpace* representation for the *Enable Method*.

Table 14 – Enable Method AddressSpace definition

Attribute	Value				
BrowseName	Enable				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
AlwaysGeneratesEvent	ObjectType	AuditConditionEnableEventType	Defined in 5.10.2		

5.5.6 AddComment Method

The *AddComment Method* is used to apply a comment to a specific state of a *Condition* instance. Normally, the *NodeId* of the *Object* instance is passed as the *ObjectId* to the *Call Service*. However, some *Servers* do not expose *Condition* instances in the *AddressSpace.* Therefore, all *Servers* shall also allow *Clients* to call the *AddComment Method* by specifying *ConditionId* as the *ObjectId.* The *Method* cannot be called with an *ObjectId* of the *ConditionType Node*.

Signature

```
AddComment(
    [in] ByteString EventId
    [in] LocalizedText Comment
    );
```

The parameters are defined in Table 15.

Table 15 – AddComment arguments

Argument	Description
EventId	EventId identifying a particular *Event Notification* where a state was reported for a *Condition*.
Comment	A localized text to be applied to the *Condition*.

Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

Method result codes in Table 16 (defined in Call Service).

Table 16 – AddComment result codes

Result Code	Description
Bad_MethodInvalid	The *MethodId* provided does not correspond to the *ObjectId* provided. See IEC 62541-4 for the general description of this result code.
Bad_EventIdUnknown	See Table 101 for the description of this result code.
Bad_NodeIdInvalid	Used to indicate that the specified *ObjectId* is not valid or that the *Method* was called on the ConditionType *Node*. See IEC 62541-4 for the general description of this result code.

Comments

Comments are added to *Event* occurrences identified via an *EventId*. *EventIds* where the related *EventType* is not a ConditionType (or subtype of it) and thus does not support *Comments* are rejected.

A *ConditionEvent* – where the *Comment Variable* contains this text – will be sent for the identified state. If a comment is added to a previous state (i.e. a state for which the Server has created a branch), the *BranchId* and all *Condition* values of this branch will be reported.

Table 17 specifies the *AddressSpace* representation for the *AddComment Method*.

Table 17 – AddComment Method AddressSpace definition

Attribute	Value				
BrowseName	AddComment				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasProperty	*Variable*	InputArguments	Argument[]	PropertyType	Mandatory
AlwaysGeneratesEvent	ObjectType	AuditConditionCommentEventType	Defined in 5.10.4		

5.5.7 ConditionRefresh Method

ConditionRefresh allows a *Client* to request a *Refresh* of all *Condition* instances that currently are in an interesting state (they have the *Retain* flag set). This includes previous states of a *Condition* instance for which the *Server* maintains *Branches*. A *Client* would typically invoke this *Method* when it initially connects to a *Server* and following any situations, such as communication disruptions, in which it would require resynchronization with the *Server*. This *Method* is only available on the *ConditionType* or its subtypes. To invoke this *Method*, the call shall pass the well-known *MethodId* of the *Method* on the *ConditionType* and the *ObjectId* shall be the well-known *ObjectId* of the *ConditionType Object*.

Signature

```
ConditionRefresh(
    [in] IntegerId SubscriptionId
    );
```


Documents
Document Sharing Hub
Full document details: DIN EN IEC 62541-9:2021-08

The parameters are defined in Table 18.

Table 18 – ConditionRefresh parameters

Argument	Description
SubscriptionId	A valid *Subscription* Id of the *Subscription* to be refreshed. The *Server* shall verify that the *SubscriptionId* provided is part of the *Session* that is invoking the *Method*.

Method result codes in Table 19 (defined in Call Service).

Table 19 – ConditionRefresh result codes

Result Code	Description
Bad_SubscriptionIdInvalid	See IEC 62541-4 for the description of this result code
Bad_RefreshInProgress	See Table 101 for the description of this result code
Bad_UserAccessDenied	The *Method* was not called in the context of the *Session* that owns the *Subscription* See IEC 62541-4 for the general description of this result code.

Comments

Subclause 4.5 describes the concept, use cases and information flow in more detail.

The input argument provides a *Subscription* identifier indicating which *Client Subscription* shall be refreshed. If the *Subscription* is accepted the *Server* will react as follows:

1) The *Server* issues an event of *RefreshStartEvent*Type(defined in 5.11.2) marking the start of *Refresh.* A copy of the instance of *RefreshStartEventType* is queued into the *Event* stream for every *Notifier MonitoredItem* in the *Subscription*. Each of the *Event* copies shall contain the same *EventId*.
2) The *Server* issues *Event Notifications* of any *Retained Conditions* and *Retained Branches* of *Conditions* that meet the *Subscriptions* content filter criteria. Note that the EventId for such a refreshed *Notification* shall be identical to the one for the original *Notification*: the values of the other *Properties* are *Server* specific, in that some *Servers* might be able to replay the exact *Events* with all *Properties*/*Variables* maintaining the same values as originally sent, but other *Servers* might only be able to regenerate the *Event*. The regenerated *Event* might contain some updated Property/*Variable* values. For example, if the *Alarm* limits associated with a *Variable* were changed after the generation of the *Event* without generating a change in the *Alarm* state, the new limit might be reported. In another example, if the HighLimit was 100 and the *Variable* is 120. If the limit were changed to 90, no new *Event* would be generated since no change to the *StateMachine*, but the limit on a *Refresh* would indicate 90, when the original *Event* had indicated 100.
3) The *Server* may intersperse new *Event Notifications* that have not been previously issued to the *Notifier* along with those being sent as part of the *Refresh* request. *Clients* shall check for multiple *Event Notifications* for a *ConditionBranch* to avoid overwriting a new state delivered together with an older state from the *Refresh* process.
4) The *Server* issues an instance of *RefreshEndEventType* (defined in 5.11.3) to signal the end of the *Refresh.* A copy of the instance of *RefreshEndEventType* is queued into the *Event* stream for every *Notifier MonitoredItem* in the *Subscription*. Each of the *Events* copies shall contain the same *EventId*.

It is important to note that if multiple *Event Notifiers* are in a *Subscription*, all *Event Notifiers* are processed. If a *Client* does not want all *MonitoredItems* refreshed, then the *Client* should place each *MonitoredItem* in a separate *Subscription* or call *ConditionRefresh2* if the *Server* supports it.

If more than one *Subscription* is to be refreshed, then the standard call *Service* array processing can be used.

Documents
Document Sharing Hub