

**DIN EN IEC 62541-9**

ICS 25.040.40; 35.200; 35.240.50

Einsprüche bis 2019-10-16  
Vorgesehen als Ersatz für  
DIN EN 62541-9:2015-11**Entwurf****OPC Unified Architecture –  
Teil 9: Alarmer und Zustände  
(IEC 65E/610/CDV:2018);  
Englische Fassung prEN IEC 62541-9:2018**OPC Unified Architecture –  
Part 9: Alarms and conditions  
(IEC 65E/610/CDV:2018);  
English version prEN IEC 62541-9:2018Architecture unifiée OPC –  
Partie 9: Alarmes et conditions  
(IEC 65E/610/CDV:2018);  
Version anglaise prEN IEC 62541-9:2018**Anwendungswarnvermerk**

Dieser Norm-Entwurf mit Erscheinungsdatum 2019-08-16 wird der Öffentlichkeit zur Prüfung und Stellungnahme vorgelegt.

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

Stellungnahmen werden erbeten

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

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

Gesamtumfang 127 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

## Nationales Vorwort

Die Englische Fassung des europäischen Dokuments prEN IEC 62541-9:2018 „OPC Unified Architecture – Teil 9: Alarime und Zustände“ (Entwurf in der Umfrage) ist unverändert in diesen Norm-Entwurf übernommen worden.

Die Internationale Elektrotechnische Kommission (IEC) und das Europäische Komitee für Elektrotechnische Normung (CENELEC) haben vereinbart, dass ein auf IEC-Ebene erarbeiteter Entwurf für eine Internationale Norm zeitgleich (parallel) bei IEC und CENELEC zur Umfrage (CDV-Stadium) und Abstimmung als FDIS (en: Final Draft International Standard) bzw. Schluss-Entwurf für eine Europäische Norm gestellt wird, um eine Beschleunigung und Straffung der Normungsarbeit zu erreichen. Dem entsprechend ist das internationale Dokument IEC 65E/610/CDV:2018 „OPC Unified Architecture – Part 9: Alarms and conditions“ unverändert in den Entwurf prEN IEC 62541-9:2018 übernommen worden.

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

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

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

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

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

Der Zusammenhang der zitierten Dokumente mit den entsprechenden deutschen Dokumenten ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

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

Da sich die Benutzer des vorliegenden Norm-Entwurfs der englischen Sprache als Fachsprache bedienen, wird die englische Fassung der prEN IEC 62541-9 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt, und Unterabschnitt 3.1, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA des vorliegenden Norm-Entwurfs hinzugefügt. Für einige der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Dieser Norm-Entwurf steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

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

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

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.

## 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

Dieses Dokument legt die Darstellung von *Alarmen* und *Bedingungen* in der OPC Unified Architecture (vereinheitlichte Architektur der plattformunabhängigen Kommunikation (OPC-UA)) fest. Darin eingeschlossen ist die Darstellung von *Alarmen* und *Bedingungen* mittels des *InformationsModells* im Adressraum des OPC-UA. Andere Aspekte von Alarmsystemen, wie etwa Alarmphilosophie, Lebenszyklus, Alarmreaktionszeiten, Alarmtypen und viele andere Einzelheiten, werden in Normen wie IEC 62682 und ISA 18.2 behandelt. Das *Alarm-* und *Bedingungen-Informationsmodell* in dieser Spezifikation wird in Übereinstimmung mit IEC 62682 und ISA 18.2 entworfen.

## 3 Begriffe und Abkürzungen

### 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 neuen *Alarms* anzeigt

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

#### 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 die *Bedingung* (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: *AdaptiveAlarmer* sind Alarmer, die durch Algorithmen automatisch angepasst werden. Diese Algorithmen können unter Umständen Bedingungen in einer Anlage erfassen und Einstellwerte oder Grenzen ändern, um zu verhindern, dass Alarmer 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 Bedingung, die als Alarmflut betrachtet werden würde, 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 *Alarmer* verwendet wird.

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

#### **BedingungsKlasse**

(en: *ConditionClass*)

Eine Gruppierung von *Bedingungen* (en: *Condition*), die anzeigt, in welcher Domäne oder für welchen Zweck eine bestimmte *Bedingung* verwendet wird

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

### 3.1.7

#### **BedingungsZweig**

(en: *ConditionBranch*)

ein bestimmter Zustand einer Bedingung

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

### 3.1.8

#### **BedingungsQuelle**

(en: *ConditionSource*)

Element, auf dem eine bestimmte *Bedingung* (en: *Condition*), basiert oder auf das sie 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.

Bei *Ereignissen* (en: *Events*), die für *Bedingungen* (en: *Condition*) erzeugt werden, enthält die *Eigenschaft* (en: *Property*) *QuellKnoten* (en: *SourceNode*) (geerbt vom *BasisEreignisTyp* (en: *BaseEventType*)) die *KnotenID* (en: *NodeId*) der *BedingungsQuelle*.

### 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

**3.1.10****Deaktivieren**

(en: Disable)

System ist 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.

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 die Prozessbedingung 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 selbsthaltenden 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. In der vorliegenden Norm 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 *Alarmer* 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 Zustand der *Bedingungen* (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

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 Alarmer 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 Alarmer geringerer Priorität für alle Ausrüstungsteile der abgeschalteten Einheit unterdrückt.

– Leerseite –

## CONTENTS

|                                              |    |
|----------------------------------------------|----|
| <b>FIGURES</b>                               | 5  |
| <b>TABLES</b>                                | 6  |
| <b>FOREWORD</b>                              | 10 |
| 1 Scope                                      | 12 |
| 2 Normative references                       | 12 |
| 3 Terms, definitions, and abbreviations      | 13 |
| 3.1 Terms and definitions                    | 13 |
| 3.2 Abbreviations and symbols                | 14 |
| 3.3 Used data types                          | 15 |
| 4 Concepts                                   | 15 |
| 4.1 General                                  | 15 |
| 4.2 Conditions                               | 15 |
| 4.3 Acknowledgeable Conditions               | 16 |
| 4.4 Previous states of Conditions            | 18 |
| 4.5 Condition state synchronization          | 18 |
| 4.6 Severity, quality, and comment           | 19 |
| 4.7 Dialogs                                  | 19 |
| 4.8 Alarms                                   | 20 |
| 4.9 Multiple active states                   | 21 |
| 4.10 Condition instances in the AddressSpace | 22 |
| 4.11 Alarm and Condition auditing            | 23 |
| 5 Model                                      | 23 |
| 5.1 General                                  | 23 |
| 5.2 Two-state state machines                 | 24 |
| 5.3 ConditionVariable                        | 25 |
| 5.4 ReferenceTypes                           | 25 |
| 5.4.1 General                                | 25 |
| 5.4.2 HasTrueSubState ReferenceType          | 26 |
| 5.4.3 HasFalseSubState ReferenceType         | 26 |
| 5.4.4 HasAlarmSuppressionGroup ReferenceType | 27 |
| 5.4.5 AlarmGroupMember ReferenceType         | 27 |
| 5.5 Condition Model                          | 27 |
| 5.5.1 General                                | 27 |
| 5.5.2 ConditionType                          | 28 |
| 5.5.3 Condition and branch instances         | 31 |
| 5.5.4 Disable Method                         | 32 |
| 5.5.5 Enable Method                          | 32 |
| 5.5.6 AddComment Method                      | 33 |
| 5.5.7 ConditionRefresh Method                | 33 |
| 5.5.8 ConditionRefresh2 Method               | 35 |
| 5.6 Dialog Model                             | 37 |
| 5.6.1 General                                | 37 |
| 5.6.2 DialogConditionType                    | 37 |
| 5.6.3 Respond Method                         | 38 |
| 5.7 Acknowledgeable Condition Model          | 39 |
| 5.7.1 General                                | 39 |

|         |                                            |    |
|---------|--------------------------------------------|----|
| 5.7.2   | AcknowledgeableConditionType.....          | 39 |
| 5.7.3   | Acknowledge Method .....                   | 40 |
| 5.7.4   | Confirm Method .....                       | 41 |
| 5.8     | Alarm model .....                          | 42 |
| 5.8.1   | General.....                               | 42 |
| 5.8.2   | AlarmConditionType.....                    | 42 |
| 5.8.3   | AlarmGroupType.....                        | 46 |
| 5.8.4   | Reset Method.....                          | 47 |
| 5.8.5   | Silence Method .....                       | 47 |
| 5.8.6   | Suppress Method .....                      | 48 |
| 5.8.7   | Unsuppress Method .....                    | 49 |
| 5.8.8   | RemoveFromService Method .....             | 49 |
| 5.8.9   | PlaceInService Method .....                | 50 |
| 5.8.10  | ShelvedStateMachineType.....               | 51 |
| 5.8.11  | LimitAlarmType .....                       | 55 |
| 5.8.12  | Exclusive Limit Types.....                 | 56 |
| 5.8.13  | NonExclusiveLimitAlarmType .....           | 59 |
| 5.8.14  | Level Alarm.....                           | 60 |
| 5.8.15  | Deviation Alarm .....                      | 61 |
| 5.8.16  | Rate of change Alarms.....                 | 62 |
| 5.8.17  | Discrete Alarms .....                      | 63 |
| 5.8.18  | DiscrepancyAlarmType .....                 | 65 |
| 5.9     | ConditionClasses .....                     | 66 |
| 5.9.1   | Overview.....                              | 66 |
| 5.9.2   | BaseConditionClassType .....               | 67 |
| 5.9.3   | ProcessConditionClassType.....             | 67 |
| 5.9.4   | MaintenanceConditionClassType .....        | 67 |
| 5.9.5   | SystemConditionClassType.....              | 67 |
| 5.9.6   | SafetyConditionClassType .....             | 68 |
| 5.9.7   | HighlyManagedAlarmConditionClassType ..... | 68 |
| 5.9.8   | TrainingConditionClassType .....           | 68 |
| 5.9.9   | StatisticalConditionClassType .....        | 69 |
| 5.9.10  | TestingConditionSubClassType.....          | 69 |
| 5.10    | Audit Events.....                          | 69 |
| 5.10.1  | Overview.....                              | 69 |
| 5.10.2  | AuditConditionEventType .....              | 70 |
| 5.10.3  | AuditConditionEnableEventType .....        | 71 |
| 5.10.4  | AuditConditionCommentEventType .....       | 71 |
| 5.10.5  | AuditConditionRespondEventType .....       | 71 |
| 5.10.6  | AuditConditionAcknowledgeEventType .....   | 71 |
| 5.10.7  | AuditConditionConfirmEventType.....        | 72 |
| 5.10.8  | AuditConditionShelvingEventType.....       | 72 |
| 5.10.9  | AuditConditionSuppressionEventType.....    | 72 |
| 5.10.10 | AuditConditionSilenceEventType.....        | 73 |
| 5.10.11 | AuditConditionResetEventType .....         | 73 |
| 5.10.12 | AuditConditionOutOfServiceEventType .....  | 73 |
| 5.11    | Condition Refresh related Events .....     | 74 |
| 5.11.1  | Overview.....                              | 74 |
| 5.11.2  | RefreshStartEventType .....                | 74 |

|                       |                                                             |    |
|-----------------------|-------------------------------------------------------------|----|
| 5.11.3                | RefreshEndEventType .....                                   | 74 |
| 5.11.4                | RefreshRequiredEventType .....                              | 74 |
| 5.12                  | HasCondition Reference type .....                           | 75 |
| 5.13                  | Alarm & Condition status codes .....                        | 75 |
| 5.14                  | Expected A&C server behaviours .....                        | 76 |
| 5.14.1                | General .....                                               | 76 |
| 5.14.2                | Communication problems .....                                | 76 |
| 5.14.3                | Redundant A&C servers .....                                 | 76 |
| 6                     | AddressSpace organisation .....                             | 77 |
| 6.1                   | General .....                                               | 77 |
| 6.2                   | EventNotifier and source hierarchy .....                    | 77 |
| 6.3                   | Adding Conditions to the hierarchy .....                    | 77 |
| 6.4                   | Conditions in InstanceDeclarations .....                    | 78 |
| 6.5                   | Conditions in a VariableType .....                          | 79 |
| 7                     | System State & Alarms .....                                 | 80 |
| 7.1                   | Overview .....                                              | 80 |
| 7.2                   | HasEffectDisable .....                                      | 80 |
| 7.3                   | HasEffectEnable .....                                       | 80 |
| 7.4                   | HasEffectSuppress .....                                     | 81 |
| 7.5                   | HasEffectUnsuppressed .....                                 | 81 |
| 8                     | Alarm Metrics .....                                         | 82 |
| 8.1                   | Overview .....                                              | 82 |
| 8.2                   | AlarmMetricsType .....                                      | 82 |
| 8.3                   | AlarmRateVariableType .....                                 | 83 |
| 8.4                   | Reset Method .....                                          | 83 |
| Annex A (informative) | Recommended localized names .....                           | 85 |
| A.1                   | Recommended state names for TwoState variables .....        | 85 |
| A.1.1                 | LocaleId "en" .....                                         | 85 |
| A.1.2                 | LocaleId "de" .....                                         | 85 |
| A.1.3                 | LocaleId "fr" .....                                         | 86 |
| A.2                   | Recommended dialog response options .....                   | 86 |
| Annex B (informative) | Examples .....                                              | 88 |
| B.1                   | Examples for Event sequences from Condition instances ..... | 88 |
| B.1.1                 | Overview .....                                              | 88 |
| B.1.2                 | Server maintains current state only .....                   | 88 |
| B.1.3                 | Server maintains previous states .....                      | 88 |
| B.2                   | AddressSpace examples .....                                 | 90 |
| Annex C (informative) | Mapping to EEMUA .....                                      | 92 |
| Annex D (informative) | Mapping from OPC A&E to OPC UA A&C .....                    | 93 |
| D.1                   | Overview .....                                              | 93 |
| D.2                   | Alarms and Events COM UA wrapper .....                      | 93 |
| D.2.1                 | Event areas .....                                           | 93 |
| D.2.2                 | Event sources .....                                         | 93 |
| D.2.3                 | Event categories .....                                      | 94 |
| D.2.4                 | Event attributes .....                                      | 95 |
| D.2.5                 | Event subscriptions .....                                   | 95 |
| D.2.6                 | Condition instances .....                                   | 97 |
| D.2.7                 | Condition Refresh .....                                     | 98 |

|         |                                         |     |
|---------|-----------------------------------------|-----|
| D.3     | Alarms and Events COM UA proxy .....    | 98  |
| D.3.1   | General .....                           | 98  |
| D.3.2   | Server status mapping .....             | 98  |
| D.3.3   | Event Type mapping .....                | 98  |
| D.3.4   | Event category mapping .....            | 99  |
| D.3.5   | Event Category attribute mapping .....  | 100 |
| D.3.6   | Event Condition mapping .....           | 103 |
| D.3.7   | Browse mapping .....                    | 103 |
| D.3.8   | Qualified names .....                   | 104 |
| D.3.9   | Subscription filters .....              | 105 |
| Annex E | – IEC62682 Mapping .....                | 107 |
| E.1     | Overview .....                          | 107 |
| E.2     | Terms .....                             | 107 |
| E.3     | Alarm records & State Indications ..... | 115 |
| Annex F | System State (Informative) .....        | 117 |
| F.1     | Overview .....                          | 117 |
| F.2     | SystemStateStateMachineType .....       | 118 |

## FIGURES

|            |                                                        |    |
|------------|--------------------------------------------------------|----|
| Figure 1   | – Base Condition state model .....                     | 16 |
| Figure 2   | – AcknowledgeableConditions state model .....          | 17 |
| Figure 3   | – Acknowledge state model .....                        | 17 |
| Figure 4   | – Confirmed Acknowledge state model .....              | 18 |
| Figure 5   | – Alarm state machine model .....                      | 20 |
| Figure 7   | – Multiple active states example .....                 | 22 |
| Figure 8   | – ConditionType hierarchy .....                        | 24 |
| Figure 9   | – Condition model .....                                | 28 |
| Figure 11  | – AcknowledgeableConditionType overview .....          | 39 |
| Figure 13  | – Alarm Model .....                                    | 43 |
| Figure 14  | – Shelve state transitions .....                       | 52 |
| Figure 15  | – ShelvedStateMachineType model .....                  | 52 |
| Figure 16  | – LimitAlarmType .....                                 | 55 |
| Figure 17  | – ExclusiveLimitStateMachineType .....                 | 57 |
| Figure 18  | – ExclusiveLimitAlarmType .....                        | 58 |
| Figure 19  | – NonExclusiveLimitAlarmType .....                     | 59 |
| Figure 20  | – DiscreteAlarmType Hierarchy .....                    | 63 |
| Figure 21  | – ConditionClass type hierarchy .....                  | 66 |
| Figure 22  | – AuditEvent hierarchy .....                           | 70 |
| Figure 23  | – Refresh Related Event Hierarchy .....                | 74 |
| Figure 24  | – Typical HasNotifier Hierarchy .....                  | 77 |
| Figure 25  | – Use of HasCondition in a HasNotifier hierarchy ..... | 78 |
| Figure 26  | – Use of HasCondition in an InstanceDeclaration .....  | 78 |
| Figure 27  | – Use of HasCondition in a VariableType .....          | 79 |
| Figure B.1 | – Single state example .....                           | 88 |

|                                                                                        |     |
|----------------------------------------------------------------------------------------|-----|
| Figure B.2 – Previous state example .....                                              | 89  |
| Figure B.3 – HasCondition used with Condition instances .....                          | 90  |
| Figure B.4 – HasCondition reference to a Condition type.....                           | 91  |
| Figure B.5 – HasCondition used with an instance declaration.....                       | 91  |
| Figure D.1 – The type model of a wrapped COM AE server .....                           | 95  |
| Figure D.2 – Mapping UA Event Types to COM A&E Event Types .....                       | 99  |
| Figure D.3 – Example mapping of UA Event Types to COM A&E categories .....             | 100 |
| Figure D.4 – Example mapping of UA Event Types to A&E categories with attributes ..... | 103 |

## TABLES

|                                                                   |    |
|-------------------------------------------------------------------|----|
| Table 1 – Parameter types defined in IEC 62541-3.....             | 15 |
| Table 2 – Parameter types defined in IEC 62541-4.....             | 15 |
| Table 3 – TwoStateVariableType definition.....                    | 24 |
| Table 4 – ConditionVariableType definition .....                  | 25 |
| Table 5 – HasTrueSubState ReferenceType.....                      | 26 |
| Table 6 – HasFalseSubState ReferenceType .....                    | 26 |
| Table 7 – HasAlarmSuppressionGroup ReferenceType .....            | 27 |
| Table 8 – AlarmGroupMember ReferenceType .....                    | 27 |
| Table 9 – ConditionType definition .....                          | 29 |
| Table 10 – SimpleAttributeOperand.....                            | 31 |
| Table 11 – Disable result codes .....                             | 32 |
| Table 12 – Disable Method AddressSpace definition .....           | 32 |
| Table 13 – Enable result codes .....                              | 32 |
| Table 14 – Enable Method AddressSpace definition .....            | 32 |
| Table 15 – AddComment arguments.....                              | 33 |
| Table 16 – AddComment result codes .....                          | 33 |
| Table 17 – AddComment Method AddressSpace definition .....        | 33 |
| Table 18 – ConditionRefresh parameters .....                      | 34 |
| Table 19 – ConditionRefresh result codes .....                    | 34 |
| Table 20 – ConditionRefresh Method AddressSpace definition .....  | 35 |
| Table 21 – ConditionRefresh2 parameters.....                      | 35 |
| Table 22 – ConditionRefresh2 result codes .....                   | 36 |
| Table 23 – ConditionRefresh2 Method AddressSpace definition ..... | 36 |
| Table 24 – DialogConditionType Definition .....                   | 37 |
| Table 25 – Respond parameters.....                                | 38 |
| Table 26 – Respond Result Codes .....                             | 38 |
| Table 27 – Respond Method AddressSpace definition .....           | 38 |
| Table 28 – AcknowledgeableConditionType definition .....          | 39 |
| Table 29 – Acknowledge parameters.....                            | 40 |
| Table 30 – Acknowledge result codes .....                         | 40 |
| Table 31 – Acknowledge Method AddressSpace definition .....       | 41 |
| Table 32 – Confirm Method parameters.....                         | 41 |

|                                                                   |    |
|-------------------------------------------------------------------|----|
| Table 33 – Confirm result codes .....                             | 41 |
| Table 34 – Confirm Method AddressSpace definition .....           | 42 |
| Table 35 – AlarmConditionType definition .....                    | 43 |
| Table 36 – AlarmGroupType Definition .....                        | 46 |
| Table 37 – Reset arguments .....                                  | 47 |
| Table 38 – Silence result codes .....                             | 47 |
| Table 39 – Reset Method AddressSpace definition .....             | 47 |
| Table 40 – Silence arguments .....                                | 48 |
| Table 41 – Silence result codes .....                             | 48 |
| Table 42 – Silence Method AddressSpace definition .....           | 48 |
| Table 43 – Suppress result codes .....                            | 48 |
| Table 44 – Suppress Method AddressSpace definition .....          | 49 |
| Table 45 – Unsuppress result codes .....                          | 49 |
| Table 46 – Unsuppress Method AddressSpace definition .....        | 49 |
| Table 47 – RemoveFromService result codes .....                   | 50 |
| Table 48 – RemoveFromService Method AddressSpace definition ..... | 50 |
| Table 49 – PlaceInService result codes .....                      | 50 |
| Table 50 – PlaceInService Method AddressSpace definition .....    | 51 |
| Table 51 –ShelvedStateMachineType definition .....                | 52 |
| Table 52 – ShelvedStateMachineType transitions .....              | 53 |
| Table 53 – Unshelve result codes .....                            | 53 |
| Table 54 – Unshelve Method AddressSpace definition .....          | 53 |
| Table 55 – TimedShelve parameters .....                           | 54 |
| Table 56 – TimedShelve result codes .....                         | 54 |
| Table 57 – TimedShelve Method AddressSpace definition .....       | 54 |
| Table 58 – OneShotShelve result codes .....                       | 55 |
| Table 59 – OneShotShelve Method AddressSpace definition .....     | 55 |
| Table 60 – LimitAlarmType definition .....                        | 56 |
| Table 61 – ExclusiveLimitStateMachineType definition .....        | 57 |
| Table 62 – ExclusiveLimitStateMachineType transitions .....       | 57 |
| Table 63 – ExclusiveLimitAlarmType definition .....               | 59 |
| Table 64 – NonExclusiveLimitAlarmType definition .....            | 60 |
| Table 65 – NonExclusiveLevelAlarmType definition .....            | 60 |
| Table 66 – ExclusiveLevelAlarmType definition .....               | 61 |
| Table 67 – NonExclusiveDeviationAlarmType definition .....        | 61 |
| Table 68 – ExclusiveDeviationAlarmType definition .....           | 62 |
| Table 69 – NonExclusiveRateOfChangeAlarmType definition .....     | 62 |
| Table 70 – ExclusiveRateOfChangeAlarmType definition .....        | 63 |
| Table 71 – DiscreteAlarmType definition .....                     | 63 |
| Table 72 – OffNormalAlarmType Definition .....                    | 64 |
| Table 73 – SystemOffNormalAlarmType definition .....              | 64 |
| Table 74 – TripAlarmType definition .....                         | 64 |
| Table 75 – InstrumentDiagnosticAlarmType definition .....         | 65 |

|                                                                  |    |
|------------------------------------------------------------------|----|
| Table 76 – SystemDiagnosticAlarmType definition .....            | 65 |
| Table 77 – CertificateExpirationAlarmType definition .....       | 65 |
| Table 78 – DiscrepancyAlarmType definition .....                 | 66 |
| Table 79 – BaseConditionClassType definition .....               | 67 |
| Table 80 – ProcessConditionClassType definition .....            | 67 |
| Table 81 – MaintenanceConditionClassType definition .....        | 67 |
| Table 82 – SystemConditionClassType definition .....             | 68 |
| Table 83 – SafetyConditionClassType definition .....             | 68 |
| Table 84 – HighlyManagedAlarmConditionClassType definition ..... | 68 |
| Table 85 – TrainingConditionClassType definition .....           | 68 |
| Table 86 – StatisticalConditionClassType definition .....        | 69 |
| Table 87 – TestingConditionSubClassType definition .....         | 69 |
| Table 88 – AuditConditionEventType definition .....              | 70 |
| Table 89 – AuditConditionEnableEventType definition .....        | 71 |
| Table 90 – AuditConditionCommentEventType definition .....       | 71 |
| Table 91 – AuditConditionRespondEventType definition .....       | 71 |
| Table 92 – AuditConditionAcknowledgeEventType definition .....   | 72 |
| Table 93 – AuditConditionConfirmEventType definition .....       | 72 |
| Table 94 – AuditConditionShelvingEventType definition .....      | 72 |
| Table 95 – AuditConditionSuppressionEventType definition .....   | 73 |
| Table 96 – AuditConditionSilenceEventType definition .....       | 73 |
| Table 97 – AuditConditionResetEventType definition .....         | 73 |
| Table 98 – AuditConditionOutOfServiceEventType definition .....  | 73 |
| Table 99 – RefreshStartEventType definition .....                | 74 |
| Table 100 – RefreshEndEventType definition .....                 | 74 |
| Table 101 – RefreshRequiredEventType definition .....            | 75 |
| Table 102 – HasCondition <i>ReferenceType</i> .....              | 75 |
| Table 103 – Alarm & Condition result codes .....                 | 76 |
| Table 104 – HasEffectDisable <i>ReferenceType</i> .....          | 80 |
| Table 105 – HasEffectEnable <i>ReferenceType</i> .....           | 81 |
| Table 106 – HasEffectSuppress <i>ReferenceType</i> .....         | 81 |
| Table 107 – HasEffectUnsuppress <i>ReferenceType</i> .....       | 82 |
| Table 108 – AlarmMetricsType Definition .....                    | 82 |
| Table 109 – AlarmRateVariableType Definition .....               | 83 |
| Table 110 – Suppress result codes .....                          | 83 |
| Table 111 – Reset Method AddressSpace definition .....           | 84 |
| Table A.1 – Recommended state names for LocaleId “en” .....      | 85 |
| Table A.2 – Recommended display names for LocaleId “en” .....    | 85 |
| Table A.3 – Recommended state names for LocaleId “de” .....      | 86 |
| Table A.4 – Recommended display names for LocaleId “de” .....    | 86 |
| Table A.5 – Recommended state names for LocaleId “fr” .....      | 86 |
| Table A.6 – Recommended display names for LocaleId “fr” .....    | 86 |
| Table A.7 – Recommended dialog response options .....            | 87 |

|                                                                                             |     |
|---------------------------------------------------------------------------------------------|-----|
| Table B.1 – Example of a Condition that only keeps the latest state .....                   | 88  |
| Table B.2 – Example of a <i>Condition</i> that maintains previous states via branches ..... | 89  |
| Table C.1 – EEMUA Terms .....                                                               | 92  |
| Table D.1 – Mapping from standard Event categories to OPC UA Event types .....              | 94  |
| Table D.2 – Mapping from ONEVENTSTRUCT fields to UA BaseEventType Variables .....           | 96  |
| Table D.3 – Mapping from ONEVENTSTRUCT fields to UA AuditEventType Variables .....          | 96  |
| Table D.4 – Mapping from ONEVENTSTRUCT fields to UA AlarmType Variables.....                | 97  |
| Table D.5 – Event category attribute mapping table .....                                    | 100 |
| Table E.1 - IEC62682 Mapping.....                                                           | 107 |
| Table F.1 – SystemStateStateMachineType definition .....                                    | 119 |
| Table F.2 - SystemStateStateMachineType transitions .....                                   | 120 |

**OPC UNIFIED ARCHITECTURE –****Part 9: Alarms & Conditions****FOREWORD**

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

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

IEC 62541-9, has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

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

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

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

- a) Added optional engineering units to the definition of RateOfChange alarms.
- b) To fulfill the IEC62682 model, the following elements have been added:

- AlarmConditionType States: Suppression, Silence, OutOfService, Latched
- AlarmConditionType Properties: OnDelay, OffDelay, FirstInGroup, RepeatTime

- New alarm types: DiscrepancyAlarm, DeviationAlarm, InstrumentDiagnosticAlarm, SystemDiagnosticAlarm
- c) Added Annex that specifies how the concepts of this OPC UA part maps to IEC 6282 and ISA 18.2.
- d) Added new ConditionClasses: Safety, HighlyManaged, Statistical, Testing, Training.
- e) Added CertificateExpiration AlarmType
- f) Added Alarm Metrics model

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

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

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

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

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

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

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

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

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

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

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

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

# OPC Unified Architecture Specification

## Part 9: Alarms & Conditions

### 1 Scope

This document 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 like alarm philosophy, life cycle, alarm response times, alarm types and many other details are captured in standards 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, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

IEC 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*

EEMUA: 2nd Edition EEMUA 191 – Alarm System – A guide to design, management and procurement (Appendixes 6, 7, 8, 9)

<https://www.eemua.org/Products/Publications/Print/EEMUA-Publication-191.aspx>

IEC 62682: Management of alarms systems for the process industries (Edition 1.0 2014-10)

<https://webstore.iec.ch/publication/7363>

ISA 18.2: Management of Alarm Systems for the Process Industries

<https://www.isa.org/store/ansi/isa-182-2016,-management-of-alarm-systems-for-the-process-industries/46962105>

IETF RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One

<https://www.ietf.org/rfc/rfc2045.txt>

IETF RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two

<https://www.ietf.org/rfc/rfc2046.txt>

IETF RFC2047: Multipurpose Internet Mail Extensions (MIME) Part Three

<https://www.ietf.org/rfc/rfc2047.txt>

### 3 Terms, definitions, and abbreviations

#### 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 as well as the following apply.

##### 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 will use *Acknowledge*.

##### 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 might 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 specification 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 may 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, it will be a *Variable* representing a process tag (e.g. FIC101) or an *Object* representing a device or subsystem.

92 In *Events* generated for *Conditions*, the *SourceNode Property* (inherited from the *BaseEventType*) will contain the  
93 *NodeId* of the *ConditionSource*.

### 94 3.1.9

#### 95 Confirm

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

### 98 3.1.10

#### 99 Disable

100 system is configured such that the *Alarm* will not be generated even though the base *Alarm*  
101 *Condition* is present

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

103 In IEC62682 disable is referenced as "Out of Service".

### 104 3.1.11

#### 105 LatchingAlarm

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

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

110

### 111 3.1.12

#### 112 Operator

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

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

### 117 3.1.13

#### 118 Refresh

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

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

### 122 3.1.14

#### 123 Retain

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

### 126 3.1.15

#### 127 Shelving

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

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

### 132 3.1.16

#### 133 Suppress

134 act of determining whether an *Alarm* should not occur

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

## 140 3.2 Abbreviations and symbols

141 A&E Alarm & Event (as used for OPC COM)

142 COM (Microsoft Windows) Component Object Model

143 DA Data Access

144 MIME Multipurpose Internet Mail Extensions

145 UA Unified Architecture

### 146 3.3 Used data types

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

150 **Table 1 – Parameter types defined in IEC 62541-3**

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

151  
152 **Table 2 – Parameter types defined in IEC 62541-4**

| Parameter Type |
|----------------|
| IntegerId      |
| StatusCode     |

## 153 154 4 Concepts

### 155 4.1 General

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

### 163 4.2 Conditions

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

- 166 • a temperature exceeding a configured limit
- 167 • a device needing maintenance
- 168 • a batch process that requires a user to confirm some step in the process before  
169 proceeding

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

176 *Condition* instances are specific implementations of a *ConditionType*. It is up to the *Server*  
177 whether such instances are also exposed in the *Server's AddressSpace*. Clause 4.10 provides  
178 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 also have to 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*. *Servers* 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. Clause 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 standard 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 sub-states.

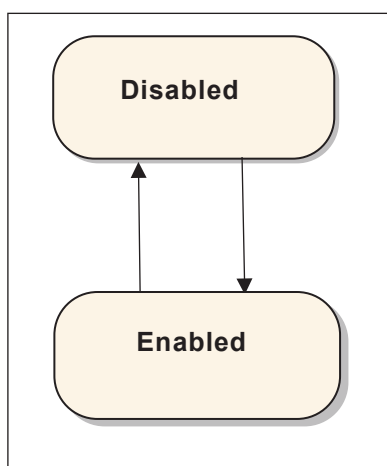


**Figure 1 – Base Condition state model**

A transition into the *Disabled* 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*.

### 4.3 Acknowledgeable Conditions

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

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

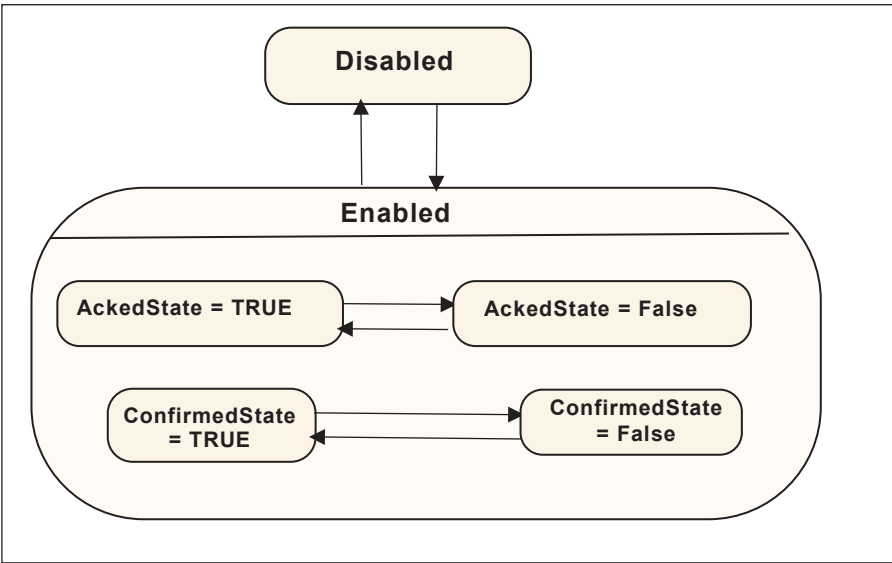


Figure 2 – AcknowledgeableConditions state model

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

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

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

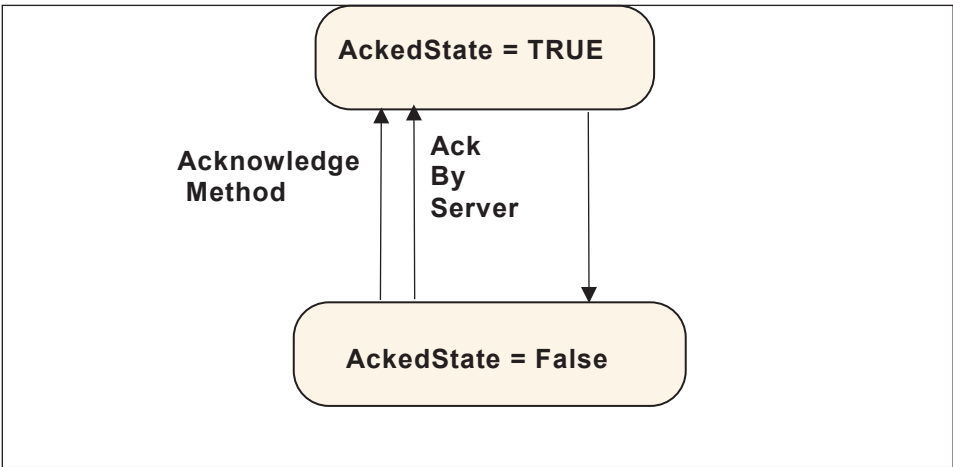


Figure 3 – Acknowledge state model

236 A more complex state model which adds a confirmation to the basic *Acknowledge* is illustrated  
237 in Figure 4. The *Confirmed Acknowledge* model is typically used to differentiate between  
238 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.

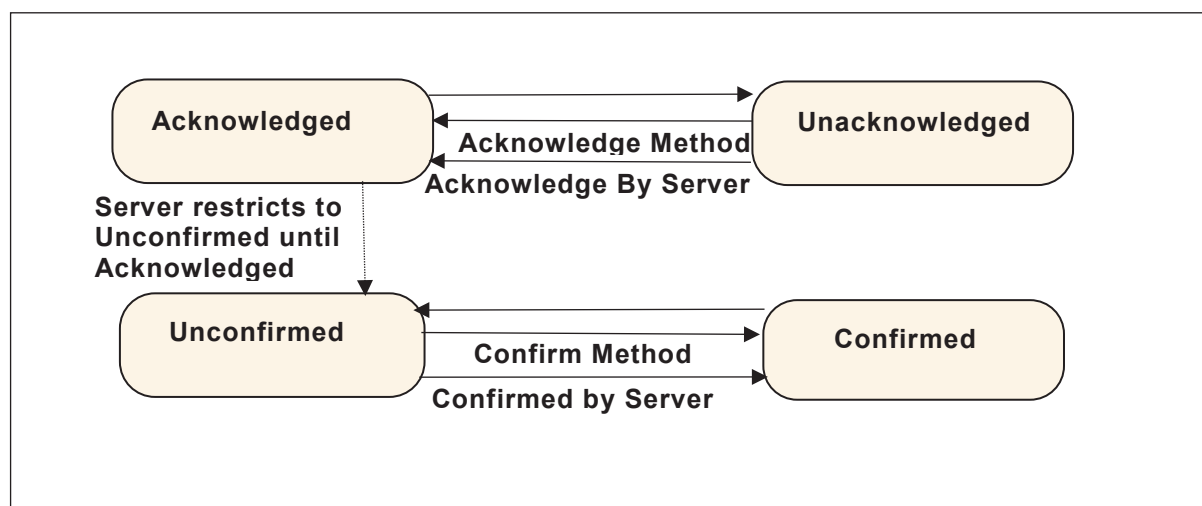


Figure 4 - Confirmed Acknowledge state model

#### 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.

#### 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 clause.
- 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 Notifications*.

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*.

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.

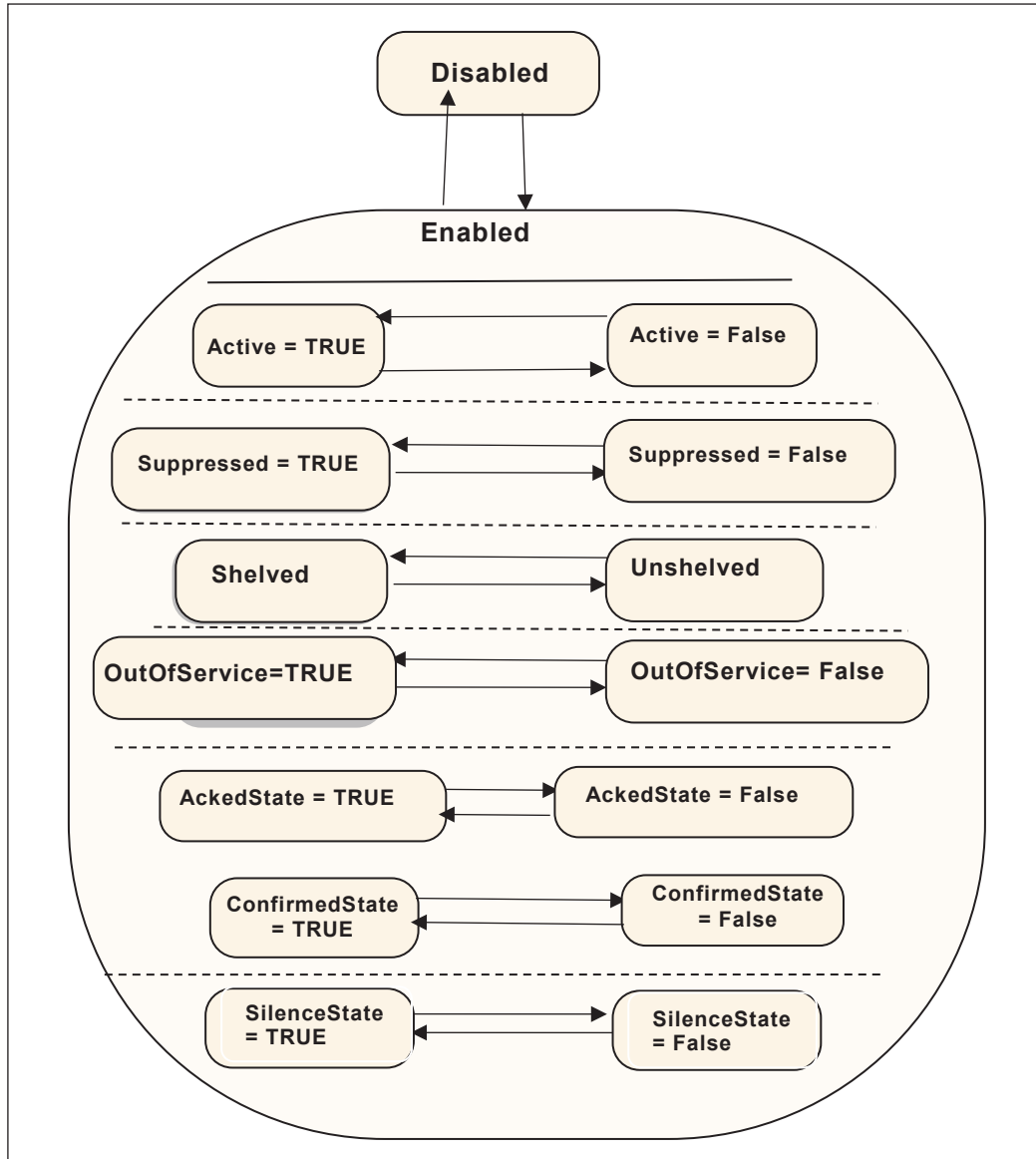


Figure 5 – Alarm state machine model

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

Some *Alarm* subtypes introduce sub-states 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 shelf 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 that 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 *Operators* about conditions in a timely manner and allow the *Operator* to take some action before some consequences occur. The consequences may be economic (product is not usable and must be discard), may be physical (tank overflows), may safety (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.

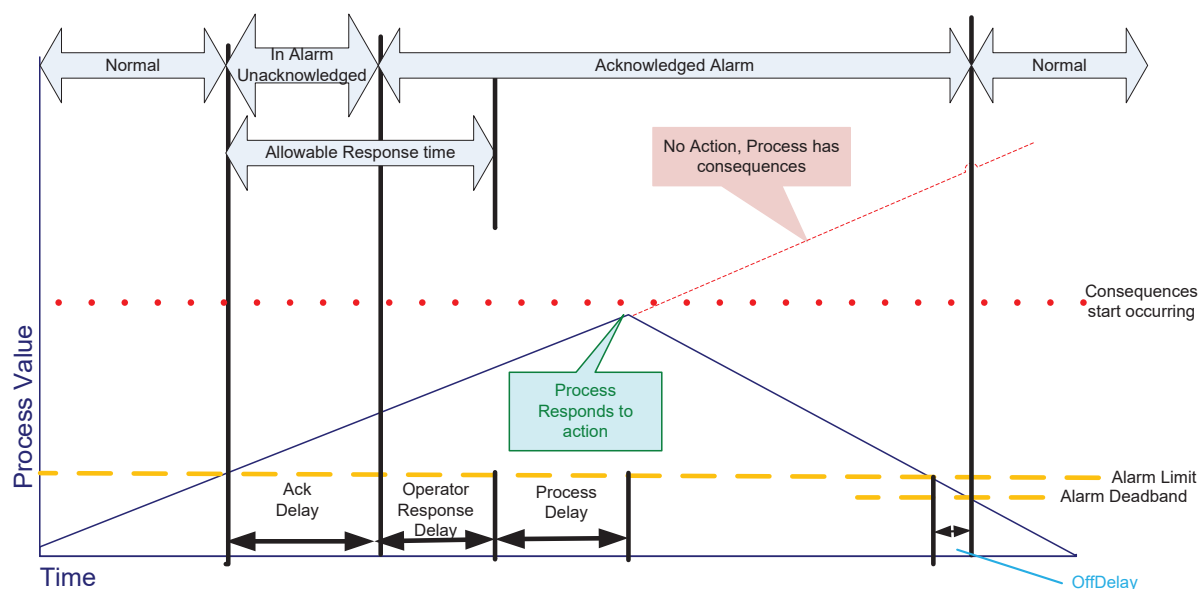


Figure 6 - Typical Alarm Timeline example

#### 4.9 Multiple active states

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

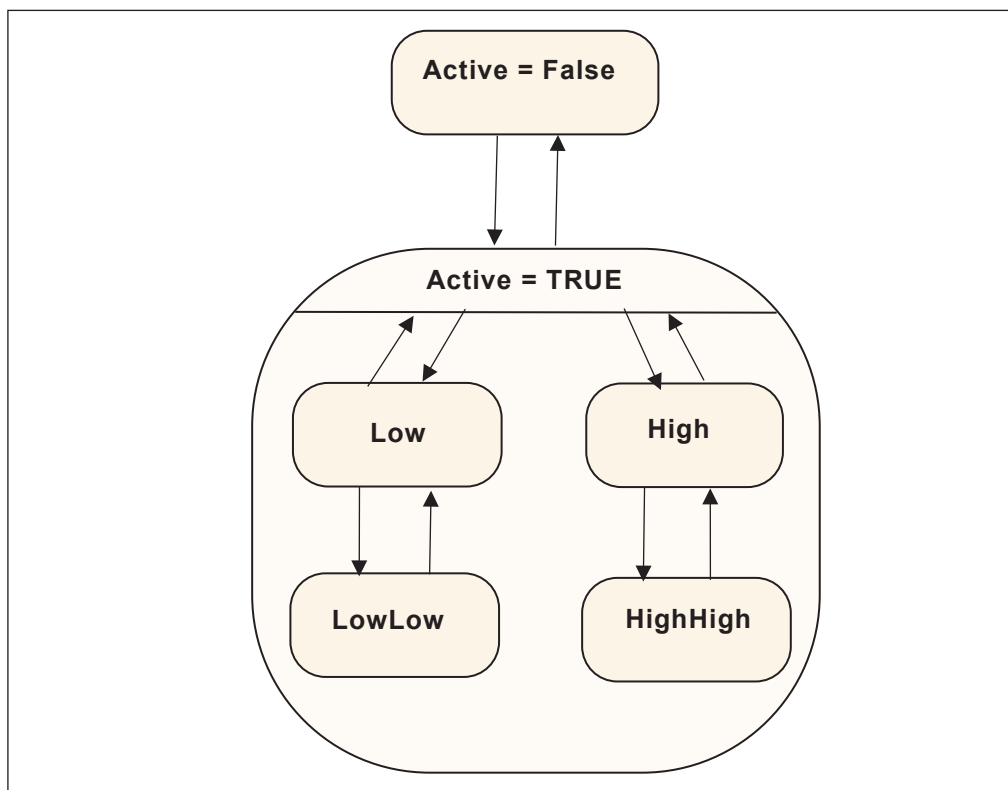


**Figure 7 – Multiple active states example**

With the multi-state *Alarm* model, state transitions among the sub-states 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 sub-state at a time. If for example a temperature exceeds the HighHigh limit the associated exclusive level *Alarm* will be in the HighHigh sub-state and not in the High sub-state.

Some *Alarm* systems, however, allow multiple sub-states 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 sub-state.

#### 4.10 Condition instances in the AddressSpace

Because *Conditions* always have a state (*Enabled* or *Disabled*) and possibly many sub-states 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.

#### 4.11 Alarm and Condition auditing

The OPC UA Standards include 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 standard defines a number of sub-types of the *AuditEventTypes* but without adding new fields to them.

This standard 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 standard 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*.

The following sub clauses 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

.

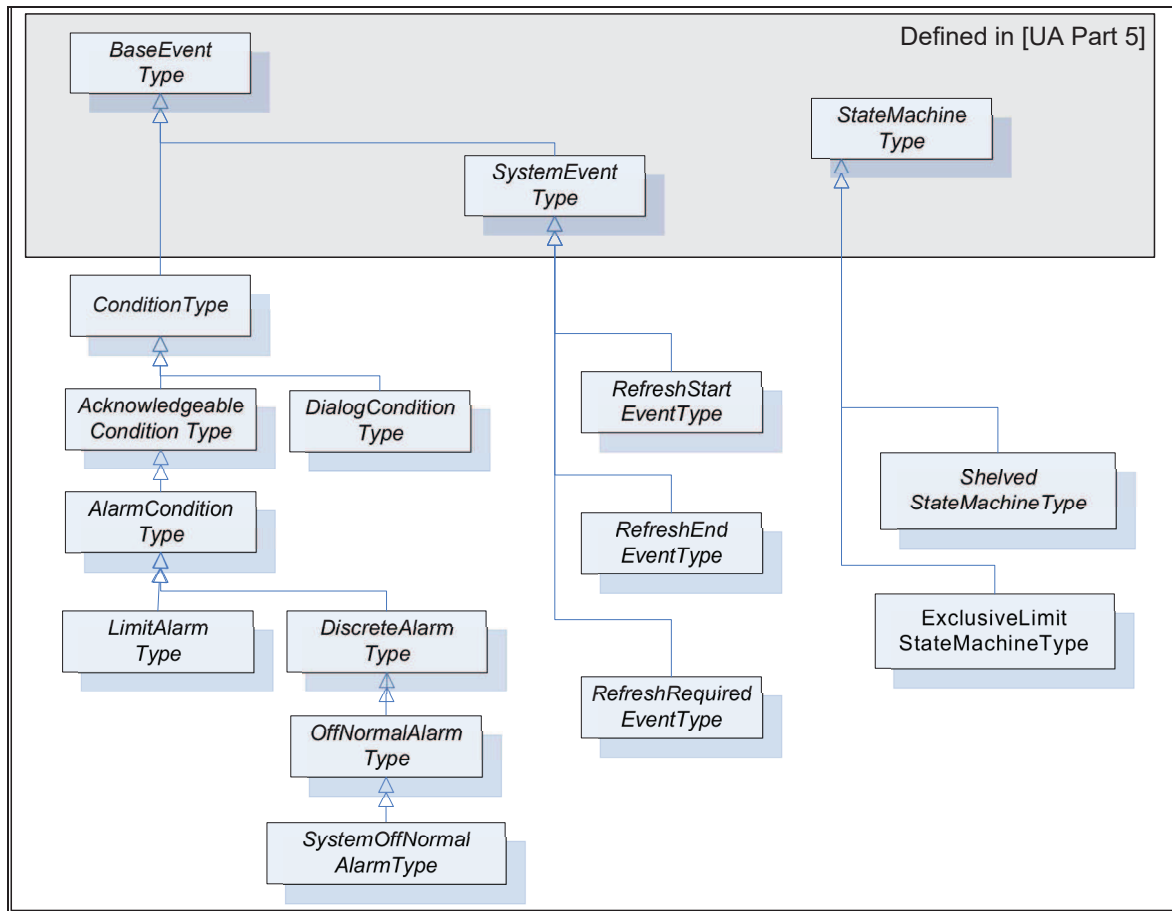


Figure 8 – ConditionType hierarchy

## 5.2 Two-state state machines

Most states defined in this standard 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.

Table 3 – TwoStateVariableType definition

| Attribute                                                                                                                                                                      | Value                                |                         |                         |                |                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-------------------------|-------------------------|----------------|----------------|
| BrowseName                                                                                                                                                                     | TwoStateVariableType                 |                         |                         |                |                |
| DataType                                                                                                                                                                       | LocalizedText                        |                         |                         |                |                |
| ValueRank                                                                                                                                                                      | -1 (-1 = Scalar)                     |                         |                         |                |                |
| IsAbstract                                                                                                                                                                     | False                                |                         |                         |                |                |
| References                                                                                                                                                                     | NodeClass                            | BrowseName              | DataType                | TypeDefinition | Modelling Rule |
| Subtype of the <i>StateVariableType</i> defined in IEC 62541-5.<br>Note that a <i>Reference</i> to this subtype is not shown in the definition of the <i>StateVariableType</i> |                                      |                         |                         |                |                |
| 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       |

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

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

437 *TransitionTime* specifies the time when the current state was entered.

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

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

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

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

455 A *HasTrueSubState Reference* is used to indicate that the True state has sub states.

456 A *HasFalseSubState Reference* is used to indicate that the False state has sub states.

### 457 5.3 ConditionVariable

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

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

462 **Table 4 – ConditionVariableType definition**

| Attribute                                                          | Value                 |                 |          |                |                |
|--------------------------------------------------------------------|-----------------------|-----------------|----------|----------------|----------------|
| BrowseName                                                         | ConditionVariableType |                 |          |                |                |
| DataType                                                           | BaseDataType          |                 |          |                |                |
| ValueRank                                                          | -2 (-2 = Any)         |                 |          |                |                |
| IsAbstract                                                         | False                 |                 |          |                |                |
| References                                                         | NodeClass             | BrowseName      | DataType | TypeDefinition | Modelling Rule |
| Subtype of the <i>BaseDataVariableType</i> defined in IEC 62541-5. |                       |                 |          |                |                |
| HasProperty                                                        | Variable              | SourceTimestamp | UtcTime  | PropertyType   | Mandatory      |

463  
 464 *SourceTimestamp* indicates the time of the last change of the *Value* of this *ConditionVariable*.  
 465 It shall be the same time that would be returned from the Read *Service* inside the *DataValue*  
 466 structure for the *ConditionVariable Value Attribute*.

### 467 5.4 ReferenceTypes

#### 468 5.4.1 General

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

The *TwoStateVariableType* *References* will only exist when sub states are available. For example, if a *TwoStateVariableType* machine is in a False State, then any sub states 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 sub state 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 sub state (the target *Node*) is a subordinate state of the True super state. If more than one state within a *Condition* is a sub state of the same super state (i.e. several *HasTrueSubState References* exist for the same super state) they are all treated as independent sub states. 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 sub state, but it is required that the sub state provides the inverse *Reference* (*IsTrueSubStateOf*) to its super state.

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 sub state (the target *Node*) is a subordinate state of the False super state. If more than one state within a *Condition* is a sub state of the same super state (i.e. several *HasFalseSubState References* exist for the same super state) they are all treated as independent sub states. 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 sub state, but it is required that the sub state 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 sub type and the *TargetNode* shall be an instance of an *AlarmGroupType*.

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 ReferenceType*.

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 sub type 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 their 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.

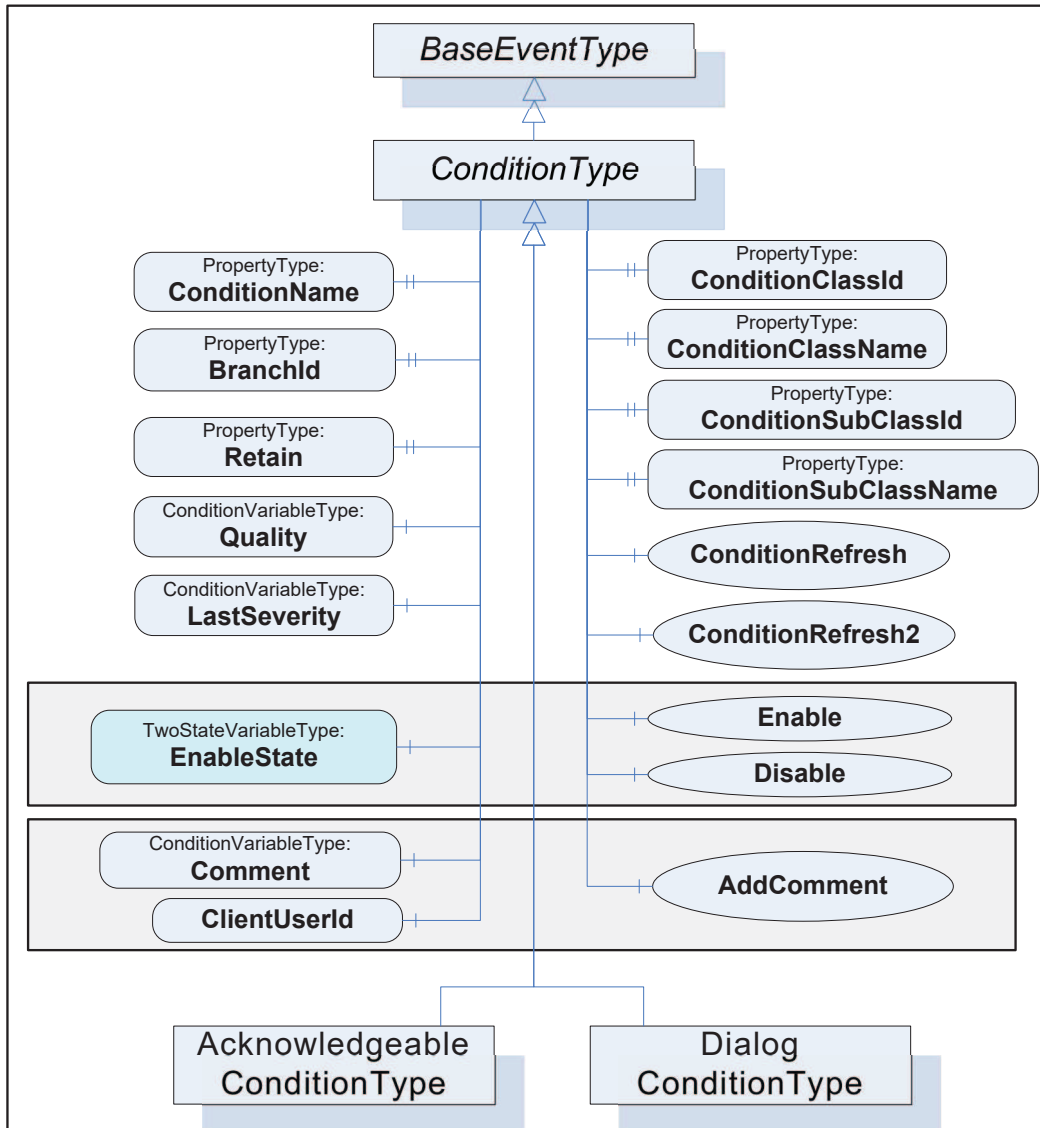


Figure 9 – Condition model

### 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 sub state machine.

542

Table 9 – ConditionType definition

| Attribute                                                  | Value         |                              |                         |                       |                |
|------------------------------------------------------------|---------------|------------------------------|-------------------------|-----------------------|----------------|
| BrowseName                                                 | ConditionType |                              |                         |                       |                |
| IsAbstract                                                 | True          |                              |                         |                       |                |
| References                                                 | NodeClass     | BrowseName                   | DataType                | TypeDefinition        | Modelling Rule |
| Subtype of the <i>BaseEventType</i> defined in IEC 62541-5 |               |                              |                         |                       |                |
| HasSubtype                                                 | ObjectType    | DialogConditionType          | Defined in Clause 5.6.2 |                       |                |
| HasSubtype                                                 | ObjectType    | AcknowledgeableConditionType | Defined in Clause 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 Clause 5.5.4 |                       | Mandatory      |
| HasComponent                                               | Method        | Enable                       | Defined in Clause 5.5.5 |                       | Mandatory      |
| HasComponent                                               | Method        | AddComment                   | Defined in Clause 5.5.6 |                       | Mandatory      |
| HasComponent                                               | Method        | ConditionRefresh             | Defined in Clause 5.5.7 |                       | None           |
| HasComponent                                               | Method        | ConditionRefresh2            | Defined in Clause 5.5.8 |                       | None           |

543

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

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

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

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

570 *ConditionSubClassName* provides the display name[s] of the *ConditionClassType[s]* listed in  
571 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 be 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* *Data Type* 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 *Clients*. 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 *Disabled*. 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 *Disabled* 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*:

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

This may not be the complete list. Sub-Types 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 <i>ConditionType</i> Node |
| browsePath[]           | QualifiedName | empty                                   |
| attributeId            | IntegerId     | Id of the <i>NodeId</i> 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*.

The state of a *Condition* instance at any given time is the set values for the *Variables* 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 standard 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 103 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*.

#### Signature

**Enable** ( ) ;

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

**Table 13 – Enable result codes**

| Result Code                 | Description                                            |
|-----------------------------|--------------------------------------------------------|
| Bad_ConditionAlreadyEnabled | See Table 103 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 |                |                |