

DIN EN IEC 62541-13

ICS 25.040.40; 35.200; 35.240.50

Ersatz für
DIN EN 62541-13:2015-10
Siehe Anwendungsbeginn

**OPC Unified Architecture –
Teil 13: Aggregation von Daten
(IEC 62541-13:2020);
Englische Fassung EN IEC 62541-13:2020**

OPC Unified Architecture –
Part 13: Aggregates
(IEC 62541-13:2020);
English version EN IEC 62541-13:2020

Architecture unifiée OPC –
Partie 13: Agrégats
(IEC 62541-13:2020);
Version anglaise EN IEC 62541-13:2020

Gesamtumfang 120 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2020-07-16 angenommene Europäische Norm als DIN-Norm ist 2021-08-01.

Für DIN EN 62541-13:2015-10 besteht eine Übergangsfrist bis 2023-07-16.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN EN IEC 62541-13:2019-09.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Da sich die Benutzer der vorliegenden Norm der englischen Sprache als Fachsprache bedienen, wird die Englische Fassung der EN IEC 62541-13 veröffentlicht. Zu deren Abschnitt 1, der den Anwendungsbereich festlegt und Abschnitt 3, der die Begriffe und Abkürzungen festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA der vorliegenden Norm hinzugefügt. Für die meisten der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Diese Norm steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

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

Änderungen

Gegenüber DIN EN 62541-13:2015-10 wurden folgende Änderungen vorgenommen:

- a) keine technischen Änderungen, aber zahlreiche Klarstellungen. Zudem einige Korrekturen der Beispiele.

Frühere Ausgaben

DIN EN 62541-13: 2015-10

Nationaler Anhang NA (informativ)

Auszugsweise Übersetzung

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

1 Anwendungsbereich

Dieser Teil von IEC 62541 ist Teil der gesamten Normenreihe zur OPC Unified Architecture (vereinheitlichte Architektur der plattformunabhängigen Kommunikation) (OPC-UA) und legt das Informationsmodell fest, welches im Zusammenhang mit der Aggregation von Daten (en: Aggregates) steht.

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 62541-11 und die folgenden Begriffe.

ISO und IEC stellen terminologische Datenbanken für die Verwendung 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

VerarbeitungsIntervall

(en: ProcessingInterval)

Zeitspanne, für die abgeleitete Werte auf der Grundlage einer bestimmten *Aggregation* (en: *Aggregate*) produziert werden

Anmerkung 1 zum Begriff: Die für Auslesungsverarbeitet (en: ReadProcessed) festgelegte gesamte Zeitdomäne wird durch das *VerarbeitungsIntervall* geteilt. Wird beispielsweise eine Durchschnittsfunktion von 10 min über einen Zeitraum von 12:00 bis 12:30 ausgeführt, ergibt dies einen Satz von drei Intervallen der Länge des *VerarbeitungsIntervalls*, wobei die einzelnen Intervalle eine Startzeit von 12:00, 12:10 bzw. 12:20 haben. Die Regeln, die zur Bestimmung der Grenzen des Intervalls verwendet werden, werden in 5.4.2.2 behandelt.

3.1.2

interpolierte Daten

(en: interpolated data)

Daten, die aus Datenstichproben berechnet werden

Anmerkung 1 zum Begriff: Datenstichproben können historische Daten oder gepufferte Echtzeitdaten sein. Ein *interpolierter* Wert wird aus den Datenpunkten vor und nach dem abgefragten Zeitstempel berechnet.

3.1.3

EffektivEndZeit

(en: EffectiveEndTime)

Zeit unmittelbar vor der *EndZeit*

Anmerkung 1 zum Begriff: Bei allen Berechnungen von *Aggregationen* (en: *Aggregate*) wird die *StartZeit* (en: *startTime*) eingeschlossen, die *EndZeit* (en: *endTime*) jedoch ausgeschlossen. Es ist jedoch manchmal notwendig, eine *interpolierte* (en: *interpolated*) Endgrenze als Wert für ein *Intervall* (en: *interval*) mit einem Zeitstempel, der innerhalb des Intervalls liegt, zurückzugeben. *Servers* müssen die Zeit unmittelbar vor der *EndZeit* verwenden, wenn die Zeitauflösung des *Servers* den

genauen Wert bestimmt (dies ist nicht mit der Zeitauflösung der Hardware oder des Betriebssystems zu verwechseln). Wenn die *EndZeit* z. B. 12:01:00 ist, entspricht die Zeitauflösung 1 s und die *EffektiveEndZeit* ist 12:00:59. Siehe 5.4.2.4.

Läuft die Zeit rückwärts, müssen *Server* die Zeit unmittelbar nach der *EndZeit* verwenden, wenn die Zeitauflösung des *Servers* den genauen Wert bestimmt.

3.1.4

extrapolierte Daten

(en: extrapolated data)

Daten, die aus einem diskreten Datensatz gebildet werden, aber außerhalb dieses Datensatzes liegen

Anmerkung 1 zum Begriff: Extrapolation ist vergleichbar mit dem Prozess der Interpolation, bei dem neue Punkte zwischen bekannten Punkten gebildet werden, das Ergebnis unterliegt jedoch einer größeren Unsicherheit. *Extrapolierte* Daten werden in Fällen verwendet, in denen der abgefragte Zeitraum weiter in der Zukunft liegt als die im zugrundeliegenden System verfügbaren Daten. Siehe Beispiel in Tabelle 1.

3.1.5

Geneigte Interpolation

(en: Sloped Interpolation)

einfache lineare Interpolation

Anmerkung 1 zum Begriff: Vergleichbar mit der Kurvenanpassung mithilfe von linearen Polynomen. Siehe Beispiel in Tabelle 1.

3.1.6

Schrittweise Interpolation

(en: Stepped Interpolation)

Interpolation durch konstant halten des letzten Datenpunkts, oder Interpolation des Werts auf der Grundlage einer horizontalen Linienanpassung

Anmerkung 1 zum Begriff: Es ist die folgende Tabelle 1 mit Rohwerten und *interpolierten/extrapolierten* (en: Interpolated/Extrapolated) Daten zu berücksichtigen:

Tabelle 1 – Beispiele für Interpolation

Zeitstempel	Rohwert	Geneigte Interpolation	Schrittweise Interpolation
12:00:00	10		
12:00:05		15	10
12:00:08		18	10
12:00:10	20		
12:00:15		25	20
12:00:20	30		
		Geneigte Extrapolation	Schrittweise Extrapolation
12:00:25		35	30
12:00:27		37	30

3.1.7

Grenzwerte

(en: Bounding Values)

Werte zur *StartZeit* und *EndZeit*, die für die *Aggregation* zur Berechnung des Ergebnisses benötigt werden

Anmerkung 1 zum Begriff: Wenn zur *StartZeit* und *EndZeit* keine Rohdaten vorhanden sind, muss ein Wert geschätzt werden. Es gibt zwei Möglichkeiten, *Grenzwerte* für ein Intervall zu bestimmen. Bei der ersten Möglichkeit (den sogenannten *interpolierten Grenzwerten*) (en: *Interpolated Bounding Values*) werden die ersten gültigen (en: non-Bad) Datenpunkte vor und nach dem Zeitstempel zur Schätzung der Grenze verwendet. Bei der zweiten Möglichkeit (den sogenannten *einfachen Grenzwerten*) (en: *Simple Bounding Values*) werden die Datenpunkte unmittelbar vor und nach

den Grenzzeitstempeln zur Schätzung der Grenze verwendet, auch wenn diese ungültig (en: Bad) sind. In 3.1.8 und 3.1.9 werden diese beiden Ansätze ausführlicher beschrieben.

In allen Fällen wird das Flag *UnsicherAlsUngültigBehandeln* (en: *TreatUncertainAsBad*) (siehe 4.2.1.2) verwendet, um festzulegen, ob unsichere Werte ungültig oder gültig sind.

Wenn kein Rohwert ermittelt wurde und ein gültiger GrenzWert vorhanden ist, werden die *Aggregat*-Bits (siehe 5.3.3) auf „interpoliert“ gesetzt.

Bei der Berechnung von *GrenzWerten* wird die Wertemenge der Rohdaten mit dem Status Ungültig auf null gesetzt. Das bedeutet, dass die Wertemenge in keiner Berechnung verwendet wird und dass im Falle der Rückgabe des Rohwertes eine Null zurückgegeben wird. Der Status wird anhand der Regeln, die durch die Grenze oder das *Aggregat* vorgegeben sind, bestimmt.

Der Ansatz der *interpolierten GrenzWerte* (siehe 3.1.8) wird auch beim klassischen OPC-Zugriff auf historische Daten angewendet und ist für Anwendungen wichtig, die jederzeit auf nützliche Werte zugreifen können müssen, z. B. in der erweiterten Prozesssteuerung. Der Ansatz der *einfachen GrenzWerte* (siehe 3.1.9) ist neu in dieser Norm und ist für Anwendungen wichtig, die regelmäßige Berichte erzeugen müssen und keine Schätzwerte für ungültige Daten verwenden können.

3.1.8

interpolierte Grenzwerte

(en: interpolated bounding values)

GrenzWerte, die durch eine Berechnung mit dem nächstliegenden gültigen Wert bestimmt werden

Anmerkung 1 zum Begriff: Interpolierte GrenzWerte, welche die GeneigteInterpolation verwenden, werden wie folgt berechnet:

- wenn zu dem Zeitstempel ein gültiger Rohwert vorhanden ist, ist dieser als GrenzWert zu verwenden;
- es ist der erste gültige Rohwert vor dem Zeitstempel zu ermitteln;
- es ist der erste gültige Rohwert nach dem Zeitstempel zu ermitteln;
- der Wert vor und der Wert nach dem Zeitstempel sind durch eine Linie zu verbinden;
- der Punkt, an dem die Linie den Zeitstempel kreuzt, ist als Schätzwert für den Grenzwert zu verwenden.

Die Berechnung kann mit der folgenden Gleichung ausgedrückt werden:

$$V_{\text{grenze}} = (T_{\text{grenze}} - T_{\text{vor}}) \times (V_{\text{nach}} - V_{\text{vor}}) / (T_{\text{nach}} - T_{\text{vor}}) + V_{\text{vor}}$$

Dabei ist V_x ein Wert bei „x“ und T_x ist der Zeitstempel im Zusammenhang mit V_x .

Wenn keine gültigen Werte vor dem Zeitstempel vorhanden sind, ist der *StatusCode* (en: *StatusCode*) *Ungültig_KeineDaten* (en: *Bad_NoData*). Wenn zwischen dem Wert vor und dem Wert nach dem Zeitstempel ungültige Werte vorhanden sind, ist der *StatusCode* *Unsicher_DatenUnterNormal* (en: *Uncertain_DataSubNormal*). Wenn entweder der Wert vor oder der Wert nach dem Zeitstempel unsicher ist, ist der *StatusCode* *Unsicher_DatenUnterNormal* (en: *Uncertain_DataSubNormal*). Wenn nach dem Zeitstempel kein Wert vorhanden ist, muss der Wert vor dem Zeitstempel unter Anwendung der *GeneigtenExtrapolation* (en: *SlopedExtrapolation*) oder der *SchrittweisenExtrapolation* (en: *SteppedExtrapolation*) extrapoliert werden.

Die Dauer der Suche nach den gültigen Werten vor und nach dem Zeitstempel ist abhängig vom *Server*; wenn innerhalb einer angemessenen Zeitspanne jedoch kein gültiger Wert ermittelt werden kann, nimmt der *Server* an, dass kein gültiger Wert vorhanden ist. Der *Server* sollte mindestens eine Zeitspanne lang suchen, die der Größe des *VerarbeitungsIntervalls* (en: *ProcessingInterval*) entspricht.

Interpolierte Grenzwerte, welche die *GeneigteExtrapolation* verwenden, werden wie folgt berechnet:

- es ist der erste gültige Rohwert vor dem Zeitstempel zu ermitteln;
- es ist der zweite gültige Rohwert vor dem Zeitstempel zu ermitteln;
- diese beiden Werte sind durch eine Linie zu verbinden;

- die Linie ist bis zu dem Punkt, an dem sie den Zeitstempel kreuzt, zu verlängern;
- der Punkt, an dem die Linie den Zeitstempel kreuzt, ist als Schätzwert für den Grenzwert zu verwenden.

Die Gleichung entspricht der Gleichung, die für die *GeneigteInterpolation* verwendet wird.

Der *StatusCode* ist immer *Unsicher_DatenUnterNormal*. Wenn vor dem Zeitstempel nur ein einziger gültiger Rohwert ermittelt werden kann, wird die *SchrittweiseExtrapolation* zur Schätzung des Grenzwertes angewendet.

Interpolierte Grenzwerte, welche die *SchrittweiseInterpolation* verwenden, werden wie folgt berechnet:

- wenn zu dem Zeitstempel ein gültiger Rohwert vorhanden ist, ist dieser als Grenzwert zu verwenden;
- es ist der erste gültige Rohwert vor dem Zeitstempel zu ermitteln;
- dieser Wert ist als Schätzwert für den Grenzwert zu verwenden.

Wenn zwischen dem Wert vor dem Zeitstempel und dem Zeitstempel ungültige Werte vorhanden sind, ist der *StatusCode* *Unsicher_DatenUnterNormal*. Wenn vor dem Zeitstempel keine gültigen Rohdaten vorhanden sind, ist der *StatusCode* *Ungültig_KeineDaten*. Wenn der Wert vor dem Zeitstempel unsicher ist, ist der *StatusCode* *Unsicher_DatenUnterNormal*. Bei der *SchrittweisenInterpolation* wird der Wert nach dem Zeitstempel nicht benötigt; liegt der Zeitstempel jedoch nach dem Ende der Daten, wird der Grenzwert als extrapoliert betrachtet und der *StatusCode* ist *Unsicher_DatenUnterNormal*.

Der Ausdruck *SchrittweiseExtrapolation* beschreibt die *SchrittweiseInterpolation* für den Fall, dass ein Zeitstempel nach dem letzten Wert in der historischen Datensammlung liegt.

3.1.9

einfache Grenzwerte

(en: simple bounding values)

Grenzwerte, die durch eine Berechnung mit dem nächstliegenden Wert bestimmt werden

Anmerkung 1 zum Begriff: *Einfache Grenzwerte*, welche die *GeneigteInterpolation* verwenden, werden wie folgt berechnet:

- wenn zu dem Zeitstempel ein beliebiger Rohwert vorhanden ist, ist dieser als Grenzwert zu verwenden;
- es ist der erste Rohwert vor dem Zeitstempel zu ermitteln;
- es ist der erste Rohwert nach dem Zeitstempel zu ermitteln;
- wenn der Wert nach dem Zeitstempel ungültig ist, wird der Wert vor dem Zeitstempel als Grenzwert verwendet;
- der Wert vor und der Wert nach dem Zeitstempel sind durch eine Linie zu verbinden;
- der Punkt, an dem die Linie den Zeitstempel kreuzt, ist als Schätzwert für den Grenzwert zu verwenden.

Die Gleichung entspricht der Gleichung, die für die *GeneigteInterpolation* in 3.1.5 verwendet wird.

Wenn ein Rohwert zu dem Zeitstempel ungültig ist, ist der *StatusCode* *Ungültig_KeineDaten*. Wenn der Wert vor dem Zeitstempel ungültig ist, ist der *StatusCode* *Ungültig_KeineDaten*. Wenn der Wert vor dem Zeitstempel unsicher ist, ist der *StatusCode* *Unsicher_DatenUnterNormal*. Wenn der Wert nach dem Zeitstempel ungültig oder unsicher ist, ist der *StatusCode* *Unsicher_DatenUnterNormal*.

Einfache Grenzwerte, welche die *SchrittweiseInterpolation* verwenden, werden wie folgt berechnet:

- wenn zu dem Zeitstempel ein beliebiger Rohwert vorhanden ist, ist dieser als Grenzwert zu verwenden;
- es ist der erste Rohwert vor dem Zeitstempel zu ermitteln;
- wenn der Wert vor dem Zeitstempel gültig ist, wird er als Grenzwert verwendet.

Wenn ein Rohwert zu dem Zeitstempel ungültig ist, ist der *StatusCode* *Ungültig_KeineDaten*. Wenn der Wert vor dem Zeitstempel ungültig ist, ist der *StatusCode* *Ungültig_KeineDaten*. Wenn der Wert vor dem Zeitstempel unsicher ist, ist der *StatusCode* *Unsicher_DatenUnterNormal*.

Wenn eine der Grenzzeiten eines Intervalls hinter dem letzten Datenpunkt liegt, kann der Server Extrapolation anwenden oder einen Fehler ausgeben. Wenn der Server Extrapolation anwendet, ist der Typ der Extrapolation [*SchrittweiseExtrapolation* oder *GeneigteExtrapolation*] serverspezifisch.

In einigen Historians^{N1} zeigt der letzte Rohwert nicht zwingend das Ende der Daten an. Auf der Grundlage der Kenntnisse des Historians über die Mechanismen zur Datensammlung, d. h. die Häufigkeit von Datenaktualisierungen und Latenzzeiten, darf der Historian den letzten Wert auf eine Zeit erweitern, die bekanntermaßen abgedeckt ist. Bei der Berechnung *einfacher Grenzwerte* verhält sich der Historian so, als ob zu diesem Zeitstempel ein weiterer Rohwert vorhanden ist.

Dasselbe gilt, wenn der früheste Zeitpunkt eines Intervalls vor dem ersten Datenpunkt in der Historie beginnt und der letzte Zeitpunkt nach dem ersten Datenpunkt in der Historie liegt; dann wird das Intervall so behandelt, als ob es sich vom ersten Datenpunkt in der Historie bis zum letzten Zeitpunkt des Intervalls erstreckt, und im *StatusCode* des Intervalls wird das Teil-Bit gesetzt (siehe 5.3.3.2).

Die Dauer der Suche nach den Werten vor und nach dem Zeitstempel ist abhängig vom *Server*, wenn innerhalb einer angemessenen Zeitspanne jedoch kein Wert ermittelt werden kann, nimmt der *Server* an, dass kein Wert vorhanden ist. Der *Server* sollte mindestens eine Zeitspanne lang suchen, die der Größe des Verarbeitungsintervalls entspricht.

3.2 Abkürzungen

DA	Datenzugriff (en: Data Access)
HA	historischer Zugriff (Zugriff auf historische Daten oder Ereignisse) (en: Historical Access (access to historical data or events))
HDA	historischer Datenzugriff (en: Historical Data Access)
UA	Unified Architecture

^{N1} Nationale Fußnote: Historian bezeichnet einen Dienst, der Daten von verschiedenen Quellen sammelt, speichert und zur Verfügung stellt.

English Version

OPC Unified Architecture - Part 13: Aggregates (IEC 62541-13:2020)

Architecture unifiée OPC - Partie 13: Agrégats
(IEC 62541-13:2020)

OPC Unified Architecture - Teil 13: Aggregation von Daten
(IEC 62541-13:2020)

This European Standard was approved by CENELEC on 2020-07-16. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

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

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

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



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

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

European foreword

The text of document 65E/697/FDIS, future edition 2 of IEC 62541-13, prepared by SC 65E "Devices and integration in enterprise systems" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN IEC 62541-13:2020.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by (dop) 2021-04-16
publication of an identical national standard or by endorsement
- latest date by which the national standards conflicting with the (dow) 2023-07-16
document have to be withdrawn

This document supersedes EN 62541-13:2015 and all of its amendments and corrigenda (if any).

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

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

Endorsement notice

The text of the International Standard IEC 62541-13:2020 was approved by CENELEC as a European Standard without any modification.

Annex ZA (normative)

Normative references to international publications with their corresponding European publications

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

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

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

<u>Publication</u>	<u>Year</u>	<u>Title</u>	<u>EN/HD</u>	<u>Year</u>
IEC/TR 62541-1	–	OPC unified architecture – Part 1: Overview and concepts	CLC/TR 62541-1	–
IEC 62541-3	–	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 62541-8	–	OPC Unified Architecture – Part 8: Data Access	–	–
IEC 62541-11	–	OPC Unified Architecture – Part 11: Historical Access	–	–

CONTENTS

European foreword	10
Annex ZA (normative) Normative references to international publications with their corresponding European publications	11
1 Scope.....	18
2 Normative references	18
3 Terms, definitions, and abbreviated terms	18
3.1 Terms and definitions	18
3.2 Abbreviated terms	22
4 Aggregate information model	22
4.1 General.....	22
4.2 Aggregate Objects.....	22
4.2.1 General.....	22
4.2.2 AggregateFunction Object.....	23
4.3 MonitoredItem AggregateFilter	25
4.3.1 MonitoredItem AggregateFilter Defaults.....	25
4.3.2 MonitoredItem Aggregates and Bounding Values.....	25
4.4 Exposing Supported Functions and Capabilities.....	26
5 Aggregate specific usage of Services	26
5.1 General.....	26
5.2 Aggregate data handling	27
5.2.1 Overview.....	27
5.2.2 ReadProcessedDetails structure overview.....	27
5.2.3 AggregateFilter structure overview.....	27
5.3 Aggregates StatusCodes.....	28
5.3.1 Overview.....	28
5.3.2 Operation level result codes	28
5.3.3 Aggregate Information Bits	29
5.4 Aggregate details.....	30
5.4.1 General.....	30
5.4.2 Common characteristics	30
5.4.3 Specific aggregated data handling	33
Annex A (informative) Aggregate Specific examples – Historical Access.....	75
A.1 Historical Aggregate specific characteristics	75
A.1.1 Example Aggregate data – Historian 1.....	75
A.1.2 Example Aggregate data – Historian 2.....	76
A.1.3 Example Aggregate data – Historian 3.....	77
A.1.4 Example Aggregate data – Historian 4.....	78
A.2 Interpolative	79

A.2.1	Description	79
A.2.2	Interpolative data	79
A.3	Average	81
A.3.1	Description	81
A.3.2	Average data	81
A.4	TimeAverage	83
A.4.1	Description	83
A.4.2	TimeAverage data	83
A.5	TimeAverage2	85
A.5.1	Description	85
A.5.2	TimeAverage2 data	85
A.6	Total	87
A.6.1	Description	87
A.6.2	Total data	87
A.7	Total2	89
A.7.1	Description	89
A.7.2	Total2 data	89
A.8	Minimum	91
A.8.1	Description	91
A.8.2	Minimum data	91
A.9	Maximum	92
A.9.1	Description	92
A.9.2	Maximum data	92
A.10	MininumActualTime	93
A.10.1	Description	93
A.10.2	MinimumActualTime data	93
A.11	MaximumActualTime	94
A.11.1	Description	94
A.11.2	MaximumActualTime data	94
A.12	Range	95
A.12.1	Description	95
A.12.2	Range data	95
A.13	Minimum2	96
A.13.1	Description	96
A.13.2	Minimum2 data	96
A.14	Maximum2	97
A.14.1	Description	97
A.14.2	Maximum2 data	97
A.15	MinimumActualTime2	98

A.15.1	Description.....	98
A.15.2	MinimumActualTime2 data	98
A.16	MaximumActualTime2	99
A.16.1	Description.....	99
A.16.2	MaximumActualTime2 data	99
A.17	Range2	100
A.17.1	Description.....	100
A.17.2	Range2 data	100
A.18	AnnotationCount.....	101
A.18.1	Description.....	101
A.18.2	AnnotationCount data.....	101
A.19	Count	101
A.19.1	Description.....	101
A.19.2	Count data	101
A.20	DurationInStateZero	103
A.20.1	Description.....	103
A.20.2	DurationInStateZero data	103
A.21	DurationInStateNonZero.....	103
A.21.1	Description.....	103
A.21.2	DurationInStateNonZero data	103
A.22	NumberOfTransitions	104
A.22.1	Description.....	104
A.22.2	NumberOfTransitions data	104
A.23	Start	105
A.23.1	Description.....	105
A.23.2	Start data	105
A.24	End	106
A.24.1	Description.....	106
A.24.2	End data	106
A.25	StartBound.....	107
A.25.1	Description.....	107
A.25.2	StartBound data.....	107
A.26	EndBound	108
A.26.1	Description.....	108
A.26.2	EndBound data	108
A.27	Delta	109
A.27.1	Description.....	109
A.27.2	Delta data	109
A.28	DeltaBounds	110

A.28.1	Description	110
A.28.2	DeltaBounds data	110
A.29	DurationGood	111
A.29.1	Description	111
A.29.2	DurationGood data	111
A.30	DurationBad	112
A.30.1	Description	112
A.30.2	DurationBad data	112
A.31	PercentGood	113
A.31.1	Description	113
A.31.2	PercentGood data	113
A.32	PercentBad	114
A.32.1	Description	114
A.32.2	PercentBad data	114
A.33	WorstQuality	115
A.33.1	Description	115
A.33.2	WorstQuality data	115
A.34	WorstQuality2	116
A.34.1	Description	116
A.34.2	WorstQuality2 data	116
A.35	StandardDeviationSample	117
A.35.1	Description	117
A.35.2	StandardDeviationSample data	117
A.36	VarianceSample	118
A.36.1	Description	118
A.36.2	VarianceSample data	118
A.37	StandardDeviationPopulation	119
A.37.1	Description	119
A.37.2	StandardDeviationPopulation data	119
A.38	VariancePopulation	120
A.38.1	Description	120
A.38.2	VariancePopulation data	120
Figure 1 – Representation of Aggregate Configuration information in the AddressSpace		26
Figure 2 – Variable with Stepped = False and Simple Bounding Values		34
Figure 3 – Variable with Stepped = True and Interpolated Bounding Values		35
Figure A.1 – Historian 1		76
Figure A.2 – Historian 2		77
Figure A.3 – Historian 3		78

Table 1 – Interpolation examples.....	19
Table 2 – AggregateConfigurationType Definition.....	22
Table 3 – Aggregate Functions Definition	23
Table 4 – AggregateFunctionType Definition	24
Table 5 – Standard AggregateType Nodes	24
Table 6 – ReadProcessedDetails	27
Table 7 – AggregateFilter structure	27
Table 8 – Bad operation level result codes	28
Table 9 – Uncertain operation level result codes	29
Table 10 – Data location	29
Table 11 – Additional information	29
Table 12 – History Aggregate interval information	31
Table 13 – Standard History Aggregate Data Type information.....	31
Table 14 – Aggregate table description.....	36
Table 15 – Interpolative Aggregate summary.....	38
Table 16 – Average Aggregate summary.....	39
Table 17 – TimeAverage Aggregate summary.....	40
Table 18 – TimeAverage2 Aggregate summary.....	41
Table 19 – Total Aggregate summary	42
Table 20 – Total2 Aggregate summary	43
Table 21 – Minimum Aggregate summary.....	44
Table 22 – Maximum Aggregate summary.....	45
Table 23 – MinimumActualTime Aggregate summary.....	46
Table 24 – MaximumActualTime Aggregate summary.....	47
Table 25 – Range Aggregate summary.....	48
Table 26 – Minimum2 Aggregate summary.....	49
Table 27 – Maximum2 Aggregate summary.....	50
Table 28 – MinimumActualTime2 Aggregate summary.....	51
Table 29 – MaximumActualTime2 Aggregate summary.....	52
Table 30 – Range2 Aggregate summary.....	53
Table 31 – AnnotationCount Aggregate summary	54
Table 32 – Count Aggregate summary.....	55
Table 33 – DurationInStateZero Aggregate summary.....	56
Table 34 – DurationInStateNonZero Aggregate Summary.....	57
Table 35 – NumberOfTransitions Aggregate summary.....	58
Table 36 – Start Aggregate summary.....	59
Table 37 – End Aggregate summary	60
Table 40 – EndBound Aggregate summary	63
Table 41 – DeltaBounds Aggregate summary.....	64

Table 42 – DurationGood Aggregate summary.....	65
Table 43 – DurationBad Aggregate summary	66
Table 44 – PercentGood Aggregate summary.....	67
Table 45 – PercentBad Aggregate summary	68
Table 46 – WorstQuality Aggregate summary	69
Table 47 – WorstQuality2 Aggregate summary	70
Table 48 – StandardDeviationSample Aggregate summary	71
Table 49 – VarianceSample Aggregate summary.....	72
Table 50 – StandardDeviationPopulation Aggregate summary	73
Table 51 – VariancePopulation Aggregate summary.....	74

OPC UNIFIED ARCHITECTURE –

Part 13: Aggregates

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture specification series and defines the information model associated with Aggregates.

2 Normative references

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

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

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

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

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

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

IEC 62541-11, *OPC Unified Architecture – Part 11: Historical Access*

3 Terms, definitions and abbreviated terms

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 62541-11 and the following apply.

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

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

3.1.1

ProcessingInterval

timespan for which derived values are produced based on a specified *Aggregate*

Note 1 to entry: The total time domain specified for ReadProcessed is divided by the *ProcessingInterval*. For example, performing a 10-minute Average over the time range 12:00 to 12:30 would result in a set of three intervals of *ProcessingInterval* length, with each interval having a start time of 12:00, 12:10 and 12:20 respectively. The rules used to determine the interval *Bounds* are discussed in 5.4.2.2.

3.1.2

interpolated data

data that is calculated from data samples

Note 1 to entry: Data samples may be historical data or buffered real time data. An *interpolated* value is calculated from the data points on either side of the requested timestamp.

3.1.3

EffectiveEndTime

time immediately before *endTime*

Note 1 to entry: All *Aggregate* calculations include the *startTime* but exclude the *endTime*. However, it is sometimes necessary to return an *Interpolated* End Bound as the value for an *Interval* with a timestamp that is in the *interval*. *Servers* are expected to use the time immediately before *endTime* where the time resolution of the *Server* determines the exact value (do not confuse this with hardware or operating system time resolution). For example, if the *endTime* is 12:01:00, the time resolution is 1 second, then the *EffectiveEndTime* is 12:00:59. See 5.4.2.4.

If time is flowing backwards, *Servers* are expected to use the time immediately after *endTime* where the time resolution of the *Server* determines the exact value.

3.1.4

extrapolated data

data constructed from a discrete data set but is outside of the discrete data set

Note 1 to entry: It is similar to the process of interpolation, which constructs new points between known points, but its result is subject to greater uncertainty. *Extrapolated* data is used in cases where the requested time period falls farther into the future than the data available in the underlying system. See example in Table 1.

3.1.5

SlopedInterpolation

simple linear interpolation

Note 1 to entry: Compare to curve fitting using linear polynomials. See example in Table 1.

3.1.6

SteppedInterpolation

interpolation holding the last data point constant or interpolating the value based on a horizontal line fit

Note 1 to entry: Consider the following Table 1 of raw and *Interpolated/Extrapolated* values:

Table 1 – Interpolation examples

Timestamp	Raw Value	Sloped Interpolation	Stepped Interpolation
12:00:00	10		
12:00:05		15	10
12:00:08		18	10
12:00:10	20		
12:00:15		25	20
12:00:20	30		
		SlopedExtrapolation	SteppedExtrapolation
12:00:25		35	30
12:00:27		37	30

3.1.7

bounding values

values at the *startTime* and *endTime* needed for *Aggregates* to compute the result

Note 1 to entry: If *Raw data* does not exist at the *startTime* and *endTime* a value shall be estimated. There are two ways to determine *Bounding Values* for an interval. One way (called *Interpolated Bounding Values*) uses the first non-Bad data points found before and after the timestamp to estimate the bound. The other (called *Simple Bounding Values*) uses the data points immediately before and after the boundary timestamps to estimate the bound even if these points are Bad. Subclauses 3.1.8 and 3.1.9 describe the two different approaches in more detail.

In all cases the *TreatUncertainAsBad* (see 4.2.1.2) flag is used to determine whether Uncertain values are Bad or non-Bad.

If a Raw value was not found and a non-Bad bounding value exists the *Aggregate* Bits (see 5.3.3) are set to 'Interpolated'.

When calculating *bounding values*, the value portion of *Raw data* that has Bad status is set to null. This means the value portion is not used in any calculation and a null is returned if the raw value is returned. The status portion is determined by the rules specified by the bound or *Aggregate*.

The *Interpolated Bounding Values* approach (see 3.1.8) is the same as what is used in Classic OPC Historical Data Access (HDA) and is important for applications such as advanced process control where having useful values at all times is important. The *Simple Bounding Values* approach (see 3.1.9) is new in this standard and is important for applications which shall produce regulatory reports and cannot use estimated values in place of Bad data.

3.1.8

interpolated bounding values

bounding values determined by a calculation using the nearest Good value

Note 1 to entry: *Interpolated Bounding Values* using *SlopedInterpolation* are calculated as follows:

- if a non-Bad Raw value exists at the timestamp then it is the bounding value;
- find the first non-Bad Raw value before the timestamp;
- find the first non-Bad Raw value after the timestamp;
- draw a line between before value and after value;
- use point where the line crosses the timestamp as an estimate of the bounding value.

The calculation can be expressed with the following formula:

$$V_{\text{bound}} = (T_{\text{bound}} - T_{\text{before}}) \times (V_{\text{after}} - V_{\text{before}}) / (T_{\text{after}} - T_{\text{before}}) + V_{\text{before}}$$

where V_x is a value at 'x' and T_x is the timestamp associated with V_x .

If no non-Bad values exist before the timestamp the *StatusCode* is *Bad_NoData*. The *StatusCode* is *Uncertain_DataSubNormal* if any Bad values exist between the before value and after value. If either the before value or the after value are Uncertain the *StatusCode* is *Uncertain_DataSubNormal*. If the after value does not exist the before value shall be extrapolated using *SlopedExtrapolation* or *SteppedExtrapolation*.

The period of time that is searched to discover the Good values before and after the timestamp is *Server* dependent, but if a Good value is not found within some reasonable time range then the *Server* will assume it does not exist. The *Server* as a minimum should search a time range which is at least the size of the *ProcessingInterval*.

Interpolated Bounding Values using *SlopedExtrapolation* are calculated as follows:

- find the first non-Bad Raw value before timestamp;
- find the second non-Bad Raw value before timestamp;
- draw a line between these two values;
- extend the line to where it crosses the timestamp;
- use the point where the line crosses the timestamp as an estimate of the bounding value.

The formula is the same as the one used for *SlopedInterpolation*.

The *StatusCode* is always *Uncertain_DataSubNormal*. If only one non-Bad raw value can be found before the timestamp then *SteppedExtrapolation* is used to estimate the bounding value.

Interpolated Bounding Values using *SteppedInterpolation* are calculated as follows:

- if a non-Bad Raw value exists at the timestamp then it is the bounding value;
- find the first non-Bad Raw value before timestamp;
- use the value as an estimate of the bounding value.

The *StatusCode* is *Uncertain_DataSubNormal* if any Bad values exist between the before value and the timestamp. If no non-Bad *Raw data* exists before the timestamp then the *StatusCode* is *Bad_NoData*. If the value before the timestamp is Uncertain the *StatusCode* is *Uncertain_DataSubNormal*. The value after the timestamp is not needed when using *SteppedInterpolation*; however, if the timestamp is after the end of the data then the bounding value is treated as extrapolated and the *StatusCode* is *Uncertain_DataSubNormal*.

SteppedExtrapolation is a term that describes *SteppedInterpolation* when a timestamp is after the last value in the history collection.

3.1.9

simple bounding values

bounding values determined by a calculation using the nearest value

Note 1 to entry: *Simple Bounding Values* using *SlopedInterpolation* are calculated as follows:

- if any Raw value exists at the timestamp then it is the bounding value;
- find the first Raw value before timestamp;
- find the first Raw value after timestamp;
- if the value after the timestamp is Bad then the before value is the bounding value;
- draw a line between before value and after value;
- use point where the line crosses the timestamp as an estimate of the bounding value.

The formula is the same as the one used for *SlopedInterpolation* in Clause 3.1.5.

If a Raw value at the timestamp is Bad the *StatusCode* is *Bad_NoData*. If the value before the timestamp is Bad the *StatusCode* is *Bad_NoData*. If the value before the timestamp is Uncertain the *StatusCode* is *Uncertain_DataSubNormal*. If the value after the timestamp is Bad or Uncertain the *StatusCode* is *Uncertain_DataSubNormal*.

Simple Bounding Values using *SteppedInterpolation* are calculated as follows:

- if any Raw value exists at the timestamp then it is the bounding value;
- find the first Raw value before timestamp;
- if the value before timestamp is non-Bad then it is the bounding value.

If a Raw value at the timestamp is Bad the *StatusCode* is *Bad_NoData*. If the value before the timestamp is Bad the *StatusCode* is *Bad_NoData*. If the value before the timestamp is Uncertain the *StatusCode* is *Uncertain_DataSubNormal*.

If either bounding time of an interval is beyond the last data point then the *Server* may use extrapolation or return an error. If extrapolation is used by the server the type [*SteppedExtrapolation* or *SloppedExtrapolation*] of extrapolation is server specific.

In some Historians, the last Raw value does not necessarily indicate the end of the data. Based on the Historian's knowledge of the data collection mechanism, i.e. frequency of data updates and latency, the Historian may extend the last value to a time known by the Historian to be covered. When calculating *Simple Bounding Values* the Historian will act as if there is another Raw value at this timestamp.

In the same way, if the earliest time of an interval starts before the first data point in history and the latest time is after the first data point in history, then the interval will be treated as if the interval extends from the first data point in history to the latest time of the interval and the *StatusCode* of the interval will have the Partial bit set (see 5.3.3.2).

The period of time that is searched to discover the values before and after the timestamp is *Server* dependent, but if a value is not found within some reasonable time range then the *Server* will assume it does not exist. The *Server* as a minimum should search a time range which is at least the size of the *ProcessingInterval*.

3.2 Abbreviated terms

DA	Data Access
HA	Historical Access (access to historical data or events)
HDA	Historical Data Access
UA	Unified Architecture

4 Aggregate information model

4.1 General

IEC 62541-3 and IEC 62541-5 standards define the representation of *Aggregate* historical or buffered real time data in the OPC Unified Architecture. This includes the definition of *Aggregates* used in processed data retrieval and in historical retrieval. This definition includes both standard *Reference* types and *Object* types.

4.2 Aggregate Objects

4.2.1 General

4.2.1.1 Overview

OPC UA *Servers* can support several different functionalities and capabilities. The following standard *Objects* are used to expose these capabilities in a common fashion, and there are several standard defined concepts that can be extended by vendors.

4.2.1.2 AggregateConfigurationType

The *AggregateConfigurationType* defines the general characteristics of a *Node* that defines the *Aggregate* configuration of any *Variable* or *Property*. *AggregateConfiguration Object* represents the browse entry point for information on how the *Server* treats *Aggregate* specific functionality such as handling Uncertain data. It is formally defined in Table 2.

Table 2 – AggregateConfigurationType Definition

Attribute	Value				
BrowseName	AggregateConfigurationType				
IsAbstract	False				
References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
Subtype of the <i>BaseObjectType</i> defined in IEC 62541-5					
HasProperty	Variable	TreatUncertainAsBad	Boolean	PropertyType	Mandatory
HasProperty	Variable	PercentDataBad	Byte	PropertyType	Mandatory
HasProperty	Variable	PercentDataGood	Byte	PropertyType	Mandatory
HasProperty	Variable	UseSlopedExtrapolation	Boolean	PropertyType	Mandatory

The *TreatUncertainAsBad Variable* indicates how the *Server* treats data returned with a *StatusCode* severity Uncertain with respect to *Aggregate* calculations. A value of True indicates the *Server* considers the severity equivalent to Bad, a value of False indicates the *Server* considers the severity equivalent to Good, unless the *Aggregate* definition says otherwise. The default value is True. Note that the value is still treated as Uncertain when the *StatusCode* for the result is calculated.

The *PercentDataBad Variable* indicates the minimum percentage of Bad data in a given interval required for the *StatusCode* for the given interval for processed data request to be set to Bad. (Uncertain is treated as defined above.) Refer to 5.4.3 for details on using this *Variable* when assigning *StatusCodes*. For details on which *Aggregates* use the *PercentDataBad Variable*, see the definition of each *Aggregate*. The default value is 100.

The *PercentDataGood Variable* indicates the minimum percentage of Good data in a given interval required for the *StatusCode* for the given interval for the processed data requests to be set to Good. Refer to 5.4.3 for details on using this *Variable* when assigning *StatusCodes*. For details on which *Aggregates* use the *PercentDataGood Variable*, see the definition of each *Aggregate*. The default value is 100.

The *PercentDataGood* and *PercentDataBad* shall follow the following relationship $PercentDataGood \geq (100 - PercentDataBad)$. If they are equal the result of the *PercentDataGood* calculation is used. If the values entered for *PercentDataGood* and *PercentDataBad* do not result in a valid calculation (e.g. Bad = 80; Good = 0) the result will have a *StatusCode* of Bad_AggregateInvalidInputs. The *StatusCode* Bad_AggregateInvalidInputs will be returned if the value of *PercentDataGood* or *PercentDataBad* exceed 100.

The *UseSlopedExtrapolation Variable* indicates how the *Server* interpolates data when no boundary value exists (i.e. extrapolating into the future from the last known value). A value of False indicates that the *Server* will use a *SteppedExtrapolation* format, and hold the last known value constant. A value of True indicates the *Server* will project the value using *UseSlopedExtrapolation* mode. The default value is False. For *SimpleBounds* this value is ignored.

4.2.2 AggregateFunction Object

4.2.2.1 General

This *Object* is used as the browse entry point for information about the *Aggregates* supported by a *Server*. The content of this *Object* is already defined by its type definition. All *Instances* of the *FolderType* use the standard *BrowseName* of 'AggregateFunctions'. The *HasComponent Reference* is used to relate a *ServerCapabilities Object* and/or any *HistoryServerCapabilitiesType Object* to an *AggregateFunction Object*. *AggregateFunctions* is formally defined in Table 3.

Table 3 – Aggregate Functions Definition

Attribute	Value				
BrowseName	AggregateFunctions				
References	Node Class	BrowseName	DataType	TypeDefinition	ModellingRule
HasTypeDefinition	Object Type	FolderType	Defined in IEC 62541-5		

Each *ServerCapabilities* and *HistoryServerCapabilitiesType Object* shall reference an *AggregateFunction Object*. In addition, each *HistoricalConfiguration Object* belonging to a *HistoricalDataNode* may reference an *AggregateFunction Object* using the *HasComponent Reference*.

4.2.2.2 AggregateFunctionType

This *ObjectType* defines an *Aggregate* supported by a UA Server. This *Object* is formally defined in Table 4.

Table 4 – AggregateFunctionType Definition

Attribute	Value				
BrowseName	AggregateFunctionType				
IsAbstract	False				
References	Node Class	BrowseName	DataType	Type Definition	Mod. Rule
Subtype of the <i>BaseObjectType</i> defined in IEC 62541-5					

For the *AggregateFunctionType*, the *Description Attribute* (inherited from the *Base NodeClass*), is mandatory. The *Description Attribute* provides a localized description of the *Aggregate*.

Table 5 specifies the *BrowseName* and *Description Attributes* for the standard *Aggregate Objects*. The description is the localized “en” text. For other locales it shall be translated.

Table 5 – Standard AggregateType Nodes

BrowseName	Description
Interpolation Aggregate	
Interpolative	At the beginning of each interval, retrieve the calculated value from the data points on either side of the requested timestamp.
Average	Retrieve the average value of the data over the interval.
TimeAverage	Retrieve the time weighted average data over the interval using <i>Interpolated Bounding Values</i> .
TimeAverage2	Retrieve the time weighted average data over the interval using <i>Simple Bounding Values</i> .
Total	Retrieve the total (time integral) of the data over the interval using <i>Interpolated Bounding Values</i> .
Total2	Retrieve the total (time integral) of the data over the interval using <i>Simple Bounding Values</i> .
Minimum	Retrieve the minimum raw value in the interval with the timestamp of the start of the interval.
Maximum	Retrieve the maximum raw value in the interval with the timestamp of the start of the interval.
MinimumActualTime	Retrieve the minimum value in the interval and the timestamp of the minimum value.
MaximumActualTime	Retrieve the maximum value in the interval and the timestamp of the maximum value.
Range	Retrieve the difference between the minimum and maximum value over the interval.
Minimum2	Retrieve the minimum value in the interval including the <i>Simple Bounding Values</i> .
Maximum2	Retrieve the maximum value in the interval including the <i>Simple Bounding Values</i> .
MinimumActualTime2	Retrieve the minimum value with the actual timestamp including the <i>Simple Bounding Values</i> .
MaximumActualTime2	Retrieve the maximum value with the actual timestamp including the <i>Simple Bounding Values</i> .
Range2	Retrieve the difference between the Minimum2 and Maximum2 value over the interval.
Count	Retrieve the number of raw values over the interval.
DurationInStateZero	Retrieve the time a Boolean or numeric was in a zero state using <i>Simple Bounding Values</i> .
DurationInStateNonZero	Retrieve the time a Boolean or numeric was in a non-zero state using <i>Simple Bounding Values</i> .

BrowseName	Description
	Interpolation Aggregate
NumberOfTransitions	Retrieve the number of changes between zero and non-zero that a Boolean or numeric value experienced in the interval.
Start	Retrieve the value at the beginning of the interval.
End	Retrieve the value at the end of the interval.
Delta	Retrieve the difference between the Start and End value in the interval.
StartBound	Retrieve the value at the beginning of the interval using <i>Simple Bounding Values</i> .
EndBound	Retrieve the value at the end of the interval using <i>Simple Bounding Values</i> .
DeltaBounds	Retrieve the difference between the StartBound and EndBound value in the interval using <i>Simple Bounding Values</i> .
DurationGood	Retrieve the total duration of time in the interval during which the data is Good.
DurationBad	Retrieve the total duration of time in the interval during which the data is Bad.
PercentGood	Retrieve the percentage of data (0 to 100) in the interval which has Good <i>StatusCode</i> .
PercentBad	Retrieve the percentage of data (0 to 100) in the interval which has Bad <i>StatusCode</i> .
WorstQuality	Retrieve the worst <i>StatusCode</i> of data in the interval.
WorstQuality2	Retrieve the worst <i>StatusCode</i> of data in the interval including the <i>Simple Bounding Values</i> .
AnnotationCount	Retrieve the number of <i>Annotations</i> in the interval (applies to Historical <i>Aggregates</i> only).
StandardDeviationSample	Retrieve the standard deviation for the interval for a sample of the population ($n-1$).
VarianceSample	Retrieve the variance for the interval as calculated by the StandardDeviationSample.
StandardDeviation Population	Retrieve the standard deviation for the interval for a complete population (n) which includes <i>Simple Bounding Values</i> .
VariancePopulation	Retrieve the variance for the interval as calculated by the StandardDeviationPopulation which includes <i>Simple Bounding Values</i> .

4.3 MonitoredItem AggregateFilter

4.3.1 MonitoredItem AggregateFilter Defaults

The default values used for *MonitoredItem Aggregates* are the same as those used for historical *Aggregates*. They are defined in 4.2.1.2. For additional information on *MonitoredItem AggregateFilter* see IEC 62541-4.

4.3.2 MonitoredItem Aggregates and Bounding Values

When calculating *MonitoredItem Aggregates* that require the use of *Bounding Values*, the bounds may not be known. The calculation is done in the same manner as a historical read with the Partial Bit set. The historian may wait some amount of time (normally no more than one processing interval) before calculating the interval to allow for any latency in data collection and reduce the use of the Partial Bit.

A historical read done after data collection and the data from the *MonitoredItem* over the same interval may not be the same.

4.4 Exposing Supported Functions and Capabilities

Figure 1 outlines a possible representation of *Aggregate* information in the *AddressSpace*. In this example, although the *Server* at the highest level may support *Aggregate* functionality for Interpolative, Total, Average, and others, *DataVariable X* only supports Interpolative, Total and Average, while *DataVariable Y* supports Average, a vendor defined *Aggregate* and other (unstated) *Aggregates*.

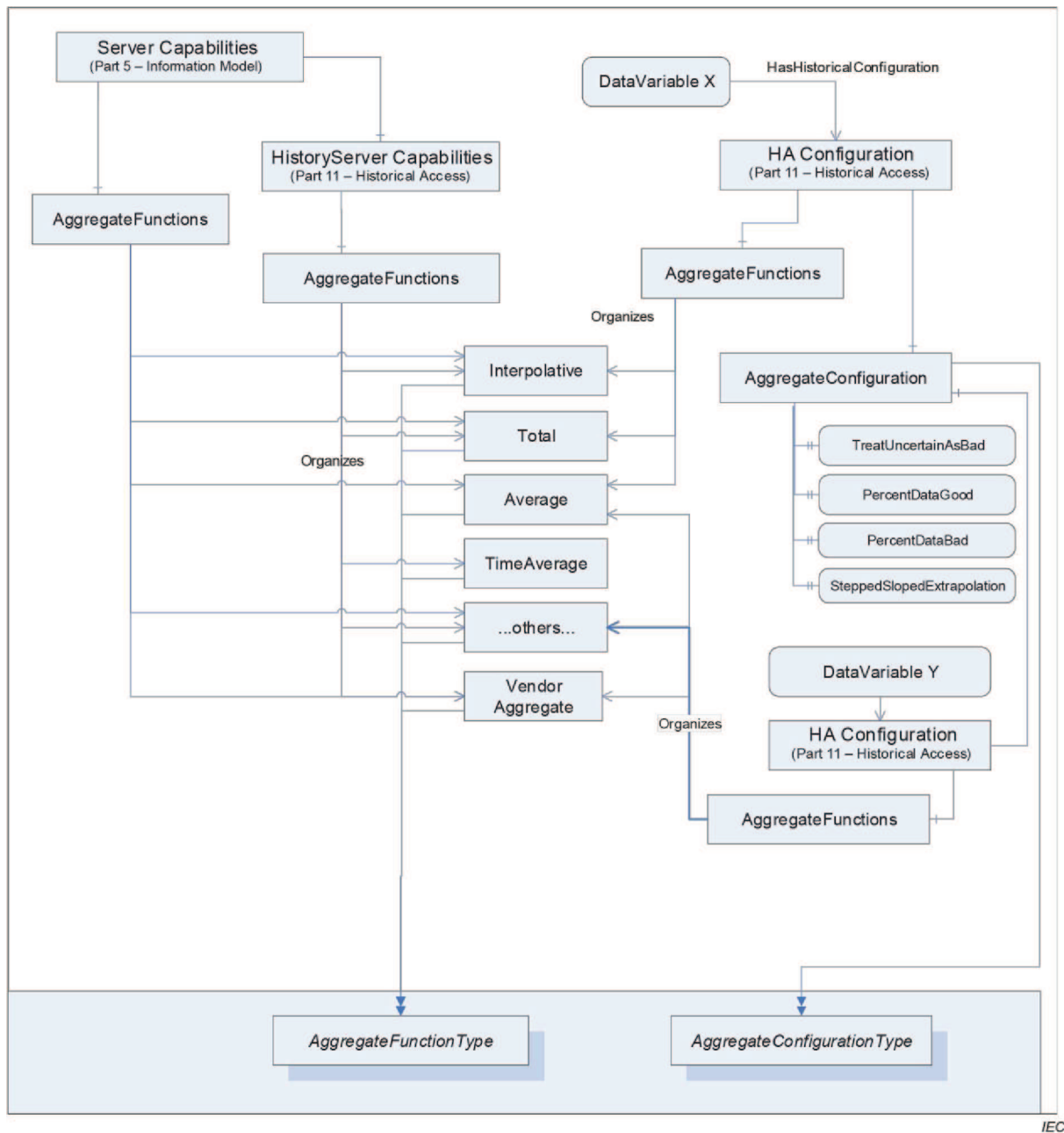


Figure 1 – Representation of Aggregate Configuration information in the AddressSpace

5 Aggregate specific usage of Services

5.1 General

IEC 62541-4 specifies all Services needed for OPC UA *Aggregates*. In particular:

- The Browse Service Set or Query Service Set to detect *Aggregates* and their configuration.

- The *HistoryRead* Service of the *Attribute Service Set* to read the aggregated history of the *HistoricalNodes*.
- The *CreateMonitoredItems* Service allows specifying a filter for each *MonitoredItem* to read aggregated data.

5.2 Aggregate data handling

5.2.1 Overview

The *HistoryRead* service defined in IEC 62541-4 can perform several different functions. The *historyReadDetails* parameter is an *Extensible Parameter* that specifies which function to perform. The *ReadProcessedDetails* structure is used to read aggregated data for *HistoricalDataNodes*.

The *CreateMonitoredItems* Service allows specifying a filter for each *MonitoredItem*. The *MonitoringFilter* is an extensible parameter whose structure depends on the type of item being monitored. The *AggregateFilter* structure is used to obtain aggregated data for a subscription.

5.2.2 ReadProcessedDetails structure overview

ReadProcessedDetails structure is formally detailed in IEC 62541-11. Table 6 outlines the components of the *ReadProcessedDetails* structure for the purposes of discussion in this document.

Table 6 – ReadProcessedDetails

Name	Description
<i>ReadProcessedDetails</i>	Specifies the details used to perform a “processed” history read.
<i>startTime</i>	Beginning of period to read.
<i>endTime</i>	End of period to read.
<i>processingInterval</i>	Interval between returned <i>Aggregate</i> values.
<i>aggregateType[]</i>	The <i>NodeIds</i> of the <i>AggregateFunction Objects</i> . <i>AggregateFunction Objects</i> indicate the list of <i>Aggregates</i> to be used when retrieving processed history.
<i>aggregateConfiguration</i>	<i>Aggregate</i> configuration structure.
<i>useServerDefaults</i>	If True the <i>Server’s</i> default values are used and any values specified for the other parameters are ignored.
<i>treatUncertainAsBad</i>	See 4.2.1.2.
<i>percentDataBad</i>	See 4.2.1.2.
<i>percentDataGood</i>	See 4.2.1.2.
<i>useSlopedExtrapolation</i>	See 4.2.1.2.

5.2.3 AggregateFilter structure overview

The *AggregateFilter* defines the *Aggregate* function that should be used to calculate the values to be returned. The *AggregateFilter* is formally defined in IEC 62541-4. Table 7 outlines the components of the *AggregateFilter* structure for the purposes of discussion in this document.

Table 7 – AggregateFilter structure

Name	Description
<i>AggregateFilter</i>	

Name	Description
startTime	Beginning of period to calculate the <i>Aggregate</i> the first time.
aggregateType	The <i>NodeIds</i> of the <i>AggregateFunction Objects</i> that indicates the list of <i>Aggregates</i> to be used when retrieving processed data.
processingInterval	The period to be used to compute the <i>Aggregate</i> .
aggregateConfiguration	This parameter allows <i>Clients</i> to override the <i>Aggregate</i> configuration settings supplied by an <i>AggregateConfiguration Object</i> on a per monitored item basis.
useServerDefaults	If True the <i>Server's</i> default values are used and any values specified for the other parameters are ignored.
treatUncertainAsBad	See 4.2.1.2.
percentDataBad	See 4.2.1.2.
percentDataGood	See 4.2.1.2.
useSlopedExtrapolation	See 4.2.1.2.

5.3 Aggregates StatusCodes

5.3.1 Overview

Subclause 5.3 defines additional codes and rules that apply to the *StatusCode* when used for *Aggregates*.

The general structure of the *StatusCode* is specified in IEC 62541-4. It includes a set of common operational result codes which also apply to *Aggregates*.

5.3.2 Operation level result codes

In OPC UA *Aggregates* the *StatusCode* is used to indicate the conditions under which a value or *Event* was stored, and thereby can be used as an indicator of its usability. Due to the nature of aggregated data, additional information beyond the basic quality and call result code needs to be conveyed to the client. For example, whether or not the result was *Interpolated*, were all data inputs to a calculation of Good quality, etc.

In the following, Table 8 contains codes with Bad severity indicating a failure; Table 9 contains codes with Uncertain severity indicating that the value has been retrieved under sub-normal conditions. It is important to note that these are the codes that are specific for OPC UA *Aggregates* and that they supplement the codes that apply to all types of data; they are therefore defined in IEC 62541-4, IEC 62541-8 and IEC 62541-11.

Table 8 – Bad operation level result codes

Symbolic Id	Description
Bad_AggregateListMismatch	The requested number of <i>Aggregates</i> does not match the requested number of <i>NodeIds</i> . When multiple <i>Aggregates</i> are requested, a corresponding <i>NodeId</i> is required for each <i>AggregateFunction</i> .
Bad_AggregateNotSupported	The requested <i>AggregateFunction</i> is not supported by the <i>Server</i> for the specified <i>Node</i> .
Bad_AggregateInvalidInputs	The <i>Aggregate</i> value could not be derived due to invalid data inputs, errors attempting to perform data conversions or similar situations.

Table 9 – Uncertain operation level result codes

Symbolic Id	Description
Uncertain_DataSubNormal	The value is derived from raw values and has less than the required number of Good values.

5.3.3 Aggregate Information Bits

5.3.3.1 General

These bits are set only when obtaining *Aggregate* data. They indicate where the data value came from and provide information that affects how the client uses the data value. Table 10 lists the bit settings which indicate the data location (i.e. is the value stored in the underlying data repository, or is the value the result of data aggregation). These bits are mutually exclusive.

Table 10 – Data location

StatusCode	Description
Raw	A <i>Raw data</i> value.
Calculated	A data value which was calculated.
Interpolated	A data value which was interpolated.

In the case where *Interpolated* data is requested, and there is an actual raw value for that timestamp, the *Server* should set the 'Raw' bit in the *StatusCode* of that value.

Table 11 lists the bit settings which indicate additional important information about the data values returned.

Table 11 – Additional information

StatusCode	Description
Partial	A calculated value that is not based on a complete interval. See 5.3.3.2.
Extra Data	If a <i>Server</i> chooses to set this bit, it indicates that a <i>Raw data</i> value supersedes other data at the same timestamp.
Multiple Values	Multiple values match the <i>Aggregate</i> criteria (i.e. multiple minimum values or multiple worst quality at different timestamps within the same <i>ProcessingInterval</i>).

The conditions under which these information bits are set depend on how the data has been requested and state of the underlying data repository.

5.3.3.2 Partial Information Bit

Partial bit is used to indicate that the interval is not a complete interval and that a client may receive a different value for the *Aggregate* if it re-fetches the interval with the same parameters.

The *Partial* bit will be set in the following examples:

Assume for these examples the first stored point in the collection is 1:01:10 and the last stored point in the collection is 1:31:20. Older data may exist but is unavailable or offline at the time of the query. Newer data may be available but has not yet been stored in the history collection.

- The interval that overlaps the beginning of the history collection. If the start time is 1:00:00 and end time is 1:10:00 and the interval is 2 minutes then the first interval would have a partial bit set since it has no data for the first 70 seconds. The *Partial* bit will always be set for the first interval with data if the start time of the interval is before the first data value of the data collection. For intervals prior to the interval with a partial bit, these intervals will be flagged *Bad_NoData*.
- The interval that overlaps the latest point stored in the history collection. The last point in the collection is 1:31:20 and the historian was not shut down and is still running. A 6-minute interval that started at 1:30:00 would have the *Partial* bit set because the historian is expecting data, but just has not yet received anything. The *Partial* bit will always be set for the last interval with data if the end time of the interval is after the last data value stored in the data collection. Intervals entirely after the interval with a *Partial* bit will be flagged *Bad_NoData*. For those *Aggregates* with extrapolation, the *Partial* bit may be set. See the *Aggregate* specific characteristics for more details.
- If the start/end time does not result in an even interval and there is additional data beyond the end time then the last interval will have a *Partial* bit. If the start time is 1:00:00 and end time is 1:20:00 and the interval is 6 minutes then the last interval is just 2 minutes long and will have the partial bit set. Extrapolation does not apply in this case.

The *Partial* bit may be set with the *Calculated* bit when the *Calculated* bit is always set for the specific *Aggregate*.

5.4 Aggregate details

5.4.1 General

The purpose of Subclause 5.4 is to detail the requirements and behaviour for OPC UA Servers supporting *Aggregates*. The intent is to standardize the *Aggregates* so users can reliably predict the results of an *Aggregate* computation and understand its meaning. If users require custom functionality in the *Aggregates*, those *Aggregates* should be written as custom vendor defined *Aggregates*.

The standard *Aggregates* shall be as consistent as possible, meaning that each *Aggregate*'s behaviour shall be similar to every other *Aggregate*'s behaviour where input parameters, *Raw data*, and boundary conditions are similar. Where possible, the *Aggregates* should deal with input and preconditions in a similar manner.

Subclause 5.4 is divided up into two parts. Subclause 5.4.2 deals with *Aggregate* characteristics and behaviour that are common to all *Aggregates*. Subclause 5.4.3 deals with the characteristics and behaviour of *Aggregates* that are aggregate-specific.

5.4.2 Common characteristics

5.4.2.1 Description

Subclause 5.4.2 deals with *Aggregate* characteristics and behaviour that are common to all *Aggregates*.

5.4.2.2 Generating intervals

To read Historical *Aggregates*, OPC clients shall specify three time parameters:

- *startTime* (Start)
- *endTime* (End)
- *ProcessingInterval* (Int)

The OPC Server shall use these three parameters to generate a sequence of time intervals and then calculate an *Aggregate* for each interval. Subclause 5.4.2.2 specifies, given the three parameters, which time intervals are generated. Table 12 provides information on the intervals for each Start and End time combination. The range is defined to be $|End - Start|$.

All *Aggregates* return a timestamp of the start of the interval unless otherwise noted for the particular *Aggregate*.

Table 12 – History Aggregate interval information

Start/End Time	Interval	Resulting intervals
$Start = End$	$Int = \text{Anything}$	No intervals. Returns a <code>Bad_InvalidArgument</code> StatusCode, regardless of whether there is data at the specified time or not.
$Start < End$	$Int = 0$ or $Int \geq Range$	One interval, starting at Start and ending at End. Includes Start, excludes End, i.e., $[Start, End)$.
$Start < End$	$Int \neq 0$, $Int < Range$, Int divides $Range$ evenly.	$Range/Int$ intervals. Intervals are $[Start, Start + Int)$, $[Start + Int, Start + 2 \times Int)$, ..., $[End - Int, End)$.
$Start < End$	$Int \neq 0$, $Int < Range$, Int does not divide $Range$ evenly.	$\lceil Range/Int \rceil$ intervals. Intervals are $[Start, Start + Int)$, $[Start + Int, Start + 2 \times Int)$, ..., $[Start + (\lceil Range/Int \rceil - 1) \times Int, Start + \lceil Range/Int \rceil \times Int)$, $[Start + \lceil Range/Int \rceil \times Int, End)$. In other words, the last interval contains the “rest” that remains in the range after taking away $\lfloor Range/Int \rfloor$ intervals of size Int .
$Start > End$	$Int = 0$ or $Int \geq Range$	One interval, starting at Start and ending at End. Includes Start, excludes End, i.e., $[Start, End)$. ^a
$Start > End$	$Int \neq 0$, $Int < Range$, Int divides $Range$ evenly.	$Range/Int$ intervals. Intervals are $[Start, Start - Int)$, $[Start - Int, Start - 2 \times Int)$, ..., $[End + Int, End)$. ^a
$Start > End$	$Int \neq 0$, $Int < Range$, Int does not divide $Range$ evenly.	$\lceil Range/Int \rceil$ intervals. Intervals are $[Start, Start - Int)$, $[Start - Int, Start - 2 \times Int)$, ..., $[Start - (\lceil Range/Int \rceil - 1) \times Int, Start - \lceil Range/Int \rceil \times Int)$, $[Start - \lceil Range/Int \rceil \times Int, End)$. In other words, the last interval contains the “rest” that remains in the range after taking away $\lfloor Range/Int \rfloor$ intervals of size Int starting at Start. ^a
^a In this case time is running backwards on the intervals.		

The calculation of all *Aggregates* when time flows backwards is the same as when time flows forwards with the exception that the ‘early time’ is excluded from the interval and the ‘late time’ is included. In most cases this means the value will be the same except the timestamps are shifted by one *ProcessingInterval*. E.g. when time flows forward the value at $T = n$ is the same as the value at $T = n + 1$ when time flows backward.

Note that when determining *Aggregates* with *MonitoredItem*, the interval is simply the *ProcessingInterval* parameter as defined in the *AggregateFilter* structure. See IEC 62541-4 for more details.

5.4.2.3 Data types

Table 13 outlines the valid *DataType* for each *Aggregate*. Some *Aggregates* are intended for numeric data types – i.e. integers or real/floating point numbers. Dates, strings, arrays, etc. are not supported. Other *Aggregates* are intended for digital data types – i.e. Boolean or enumerations. In addition some *Aggregates* may return results with a different *DataType* than those used to calculate the *Aggregate*. Table 13 also outlines the data type returned for each *Aggregate*.

Table 13 – Standard History Aggregate Data Type information

BrowseName	Valid Data Type	Result Data Type
	Interpolation Aggregate	
Interpolative	Numeric	Raw Data Type
	Data Averaging Aggregates	

BrowseName	Valid Data Type	Result Data Type
Average	Numeric	Double
TimeAverage	Numeric	Double
TimeAverage2	Numeric	Double
Total	Numeric	Double
Total2	Numeric	Double
	Data Variation Aggregates	
Minimum	Numeric	Raw data type
Maximum	Numeric	Raw data type
MinimumActualTime	Numeric	Raw data type
MaximumActualTime	Numeric	Raw data type
Range	Numeric	Raw data type
Minimum2	Numeric	Raw data type
Maximum2	Numeric	Raw data type
MinimumActualTime2	Numeric	Raw data type
MaximumActualTime2	Numeric	Raw data type
Range2	Numeric	Raw data type
	Counting Aggregates	
AnnotationCount	All	Integer
Count	All	Integer
DurationInStateZero	Numeric or Boolean	Duration
DurationInStateNonZero	Numeric or Boolean	Duration
NumberOfTransitions	Numeric or Boolean	Integer
	Time Aggregates	
Start	All	Raw data type
End	All	Raw data type
Delta	Numeric	Raw data type
StartBound	All	Raw data type
EndBound	All	Raw data type
DeltaBounds	Numeric	Raw data type
	Data Quality Aggregates	
DurationGood	All	Duration
DurationBad	All	Duration
PercentGood	All	Double
PercentBad	All	Double
WorstQuality	All	StatusCode
WorstQuality2	All	StatusCode
	Statistical Aggregates	

BrowseName	Valid Data Type	Result Data Type
StandardDeviationSample	Numeric	Double
VarianceSample	Numeric	Double
StandardDeviationPopulation	Numeric	Double
VariancePopulation	Numeric	Double

5.4.2.4 Time calculation issues

The following issues may come up when calculating *Aggregates* that include time as part of the calculation.

- All *Aggregate* calculations include the *startTime* but exclude the *endTime*. However, it is sometimes necessary to return an *Interpolated* End Bound as the value for an *Interval* with a timestamp that is in the *Interval*. *Servers* are expected to use the time immediately before *endTime* where the time resolution of the *Server* determines the exact value (do not confuse this with hardware or operating system time resolution). For example, if the *endTime* is 12:01:00, the time resolution is 1 second, then the *EffectiveEndTime* is 12:00:59. If the *Server* time resolution is 1 millisecond the *EffectiveEndTime* is 12:00:59.999.
- If time is flowing backwards, *Servers* are expected to use the time immediately after *endTime* where the time resolution of the *Server* determines the exact value.
- If there is one data point in the *Interval* and it falls on the *StartTime* the time duration used in calculations is one unit of the time resolution of the *Server*.

5.4.3 Specific aggregated data handling

5.4.3.1 General

When accessing aggregated data using the *HistoryRead* or the *CreateMonitoredItems Service*, the following rules are used to handle specific *Aggregate* use cases.

If *ProcessingInterval* is 0, the *Server* shall create one *Aggregate* value for the entire time range. This allows *Aggregates* over large periods of time. A value with a timestamp equal to *endTime* will be excluded from that *Aggregate*, just as it would be excluded from an interval with that ending time. If the *ProcessingInterval* of 0 is passed in the *MonitoredItemFilter* it shall be revised to a suitable non-zero value.

The timestamp returned with the *Aggregate* shall be the time at the beginning of the interval, except where the *Aggregate* specifies a different timestamp.

If a requested timestamp is set to anything but the source timestamp the operation shall return the *Bad_TimestampToReturnInvalid StatusCode*. If a requested timestamp is not supported in any other way for a *HistoricalDataNode*, the operation shall return the *Bad_TimestampNotSupported StatusCode*. For *MonitoredItem* the *Server* shall not return past data if a requested timestamp is not supported by the history collection.

5.4.3.2 StatusCode calculation

5.4.3.2.1 General

StatusCodes for an *Aggregate* value shall take into account the values used to calculate them. In addition, the configuration parameters *PercentDataGood* and *PercentDataBad* allow the client to control how this calculation is done if supported by the *Server*.

If an *Aggregate* operates on raw values (e.g. Average) the calculation is done by counting values. If an *Aggregate* operates on raw values but can also return a *Bounding Value* then the *Bounding Values* are included

in the count when computing the *StatusCode*. If an *Aggregate* does any sort of a time weighted calculation (e.g. *TimeAverage* or *TimeAverage2*) then the *StatusCode* calculation shall also be time weighted.

For purposes of calculating time weighted *StatusCodes* each interval shall be divided into regions of Good or Bad data. Creating these regions requires that the *bounding values* be calculated for each interval and the type of bounding value depends on the *Aggregate*.

If *TreatUncertainAsBad* = False then Uncertain regions are included with the Good regions when calculating the above ratios, if the *TreatUncertainAsBad* = True then the Uncertain regions are included as Bad regions. The *StatusCode* of the value is still treated as Uncertain when the *StatusCode* for the result is calculated. If no Bad regions are in the interval then the *StatusCode* for the interval is Good. For any intervals containing regions where the *StatusCodes* are Bad, the total duration of all *Bad* regions is calculated and divided by the width of the interval. The resulting ratio is multiplied by 100 and compared to the *PercentDataBad* parameter. The *StatusCode* for the interval is Bad if the ratio is greater than or equal to the *PercentDataBad* parameter. For any interval which is not Bad, the total duration of all Good regions is then calculated and divided by the width of the interval. The resulting ratio is multiplied by 100 and compared to the *PercentDataGood* parameter. The *StatusCode* for the interval is Good if the ratio is greater than or equal to the *PercentDataGood* parameter. If for an interval neither ratio applies then that interval is *Uncertain_DataSubNormal*.

If there is no data in the interval and the interval is inside the range [start of data, end of data] and the *Aggregate* return data type is raw data type then the *StatusCodes* for the interval will be *Bad_NoData* unless an alternate status code is defined for a specific *Aggregate*.

The width of an interval is the *ProcessingInterval* unless it is a partial interval (i.e. has the Partial bit set). In these cases, the width is the time used when calculating the partial interval.

Subclauses 5.4.3.2.2 and 5.4.3.2.3 include diagrams that illustrate a request and data series. The colour of the time axis indicates the status for different regions. Red indicates Bad, green indicates Good and orange indicates Uncertain. These examples assume *TreatUncertainAsBad* = False.

5.4.3.2.2 Sloped Interpolation and Simple Bounding Values

Figure 2 illustrates a data series for *Variable* with *Stepped* = False and an *Aggregate* that uses *Simple Bounding Values*. The request being processed has a Start Time that falls before the first point in the series and an End Time that does not fall on an integer multiple of the *ProcessingInterval*.

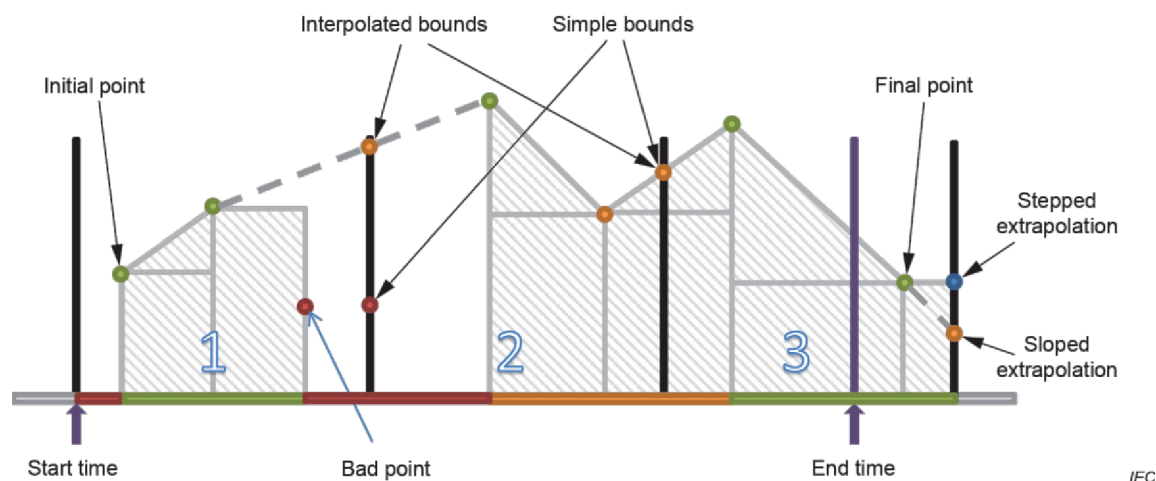


Figure 2 – Variable with Stepped = False and Simple Bounding Values

The first interval has four regions:

- the period before the first data point;
- the period between the first and second where *SlopedInterpolation* can be used;

- the period between the second and third point where *SteppedInterpolation* is used;
- the period after the Bad point where no data exists.

A region is Uncertain if a region ends in a Bad or Uncertain value and *SlopedInterpolation* is used. The end point has no effect on the region if *SteppedInterpolation* is used.

The second interval has three regions:

- the period before the first Good data point where no data exists;
- the period between the first and second where *SlopedInterpolation* can be used;
- the period between the second point and the bound calculated with *SlopedInterpolation*.

The third interval has three regions:

- the period between the simple bound and the first data point;
- the period between the first point and an interpolated point that falls on the end time;
- the period after the end time which is ignored.

This is a partial region and the data after the end time is not used, however, if sloped interpolation is used and the point after the endpoint is Uncertain then the region between the last point and the end time will be Uncertain.

5.4.3.2.3 Stepped Interpolation and Interpolated Bounding Values

Figure 3 illustrates a data series for *Variable* with *Stepped* = True and an *Aggregate* that uses *Interpolated Bounding Values*. The request being processed has a Start Time that falls before the first point in the series and an End Time that does not fall on an integer multiple of the *ProcessingInterval*.

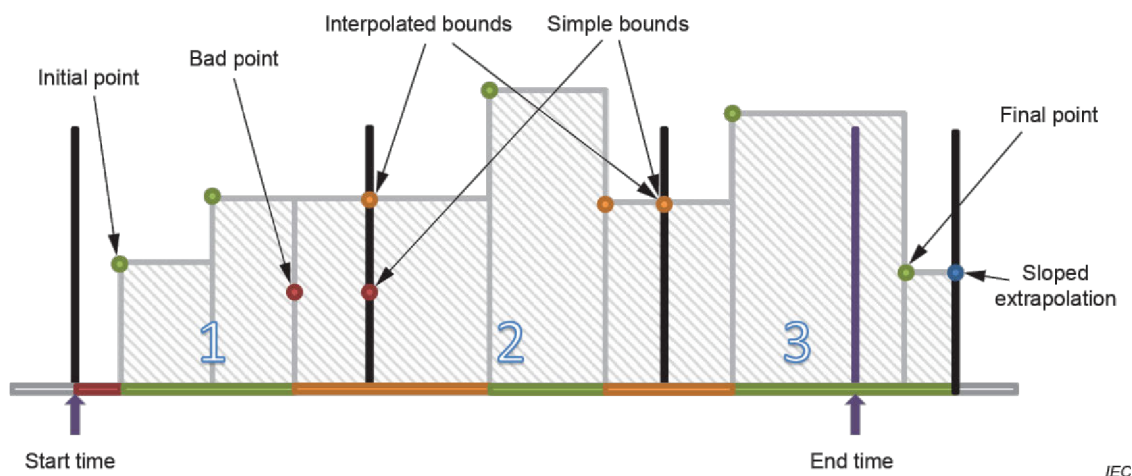


Figure 3 – Variable with *Stepped* = True and Interpolated Bounding Values

The first interval has three regions:

- the period before the first data point;
- the period between the first and second where *SteppedInterpolation* is used;
- the period between the second and the interpolated end bound.

The Bad point is ignored because of the interpolated end bound but this does create Uncertain regions. If *SlopedInterpolation* was used the Uncertain region would start at the second point. In this case, it only starts when the first Bad value is ignored.

The second interval has three regions:

- the period between the start bound and the first data point;
- the period between the first and second where *SteppedInterpolation* is used;
- the period between the second and the interpolated end bound.

The third interval has three regions:

- the period between the interpolated bound and the first data point;
- the period between the first point and an interpolated point that falls on the end time;
- the period after the end time which is ignored.

This is a partial region and the data after the end time is not used.

5.4.3.3 Description

Subclause 5.4.3.3 deals with *Aggregate* specific characteristics and behaviour that is specific to a particular *Aggregate*.

Each subclause has a table which formally expresses the *Aggregate* behaviour (including any exceptions). The meaning of each of the fields in the table is described in Table 14.

Description of Table 14:

- The first column is the common name for the item.
- The second column includes a description of the item and a list of the valid selections with for the item including a description of each selection.
- The second part of the table describes how the status associated with the *Aggregate* calculation is computed.
- The last part of the table lists what behaviour is expected from the *Aggregate* for some common special cases. These behaviours require text descriptions so there is no list of valid selections.

Table 14 – Aggregate table description

Aggregate Characteristics	
Type	<p>The type of <i>Aggregate</i>.</p> <p><Interpolated Calculated Raw></p> <p>Interpolated: See definition for Interpolated.</p> <p>Calculated: Computed from defined calculation.</p> <p>Raw: Selects a raw value from within an interval.</p>
Data Type	<p>The data type of the result.</p> <p><Double Int32 Same as Source></p>
Use Bounds	<p>How the <i>Aggregate</i> deals with bounds.</p> <p><None Interpolated Simple></p> <p>None: Bounds do not apply to the <i>Aggregate</i>.</p> <p>Interpolated: Uses Interpolated Bounds.</p> <p>Simple: Uses Simple Bounds.</p>
Timestamp	<p>What is the time stamp of the resulting <i>Aggregate</i> value:</p>