IEC 62541-11

Edition 1.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 11: Historical Access

Architecture unifiée OPC – Partie 11: Accès à l'Historique

IEC 62541-11:2015-03(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 15 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 60 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 62541-11

Edition 1.0 2015-03

INTERNATIONAL STANDARD

NORME INTERNATIONALE

colour inside

OPC unified architecture – Part 11: Historical Access

Architecture unifiée OPC – Partie 11: Accès à l'Historique

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100

ISBN 978-2-8322-2298-0

Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

OPC UNIFIED ARCHITECTURE –

Part 11: Historical Access

FOREWORD

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

International Standard IEC 62541-11 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this standard is based on the following documents:

CDV	Report on voting
65E/380/CDV	65E/410/RVC

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

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

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

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.

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 –

Part 11: Historical Access

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture standard series and defines the *information model* associated with Historical Access (HA). It particularly includes additional and complementary descriptions of the *NodeClass*es and *Attributes* needed for Historical Access, additional standard *Properties*, and other information and behaviour.

The complete *AddressSpace* Model including all *NodeClass*es and *Attributes* is specified in IEC 62541-3. The predefined *Information Model* is defined in IEC 62541-5. The *Services* to detect and access historical data and events, and description of the *ExtensibleParameter* types are specified in IEC 62541-4.

This standard includes functionality to compute and return *Aggregates* like minimum, maximum, average etc. The *Information Model* and the concrete working of *Aggregates* are defined in IEC 62541-13.

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

IEC 62541-13, *OPC Unified Architecture – Part 13: Aggregates*

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 62541-13 as well as the following apply.

3.1.1
Annotation
metadata associated with an item at a given instance in time

Note 1 to entry: An *Annotation* is metadata that is associated with an item at a given instance in time. There does not have to be a value stored at that time.

3.1.2
BoundingValues
values associated with the starting and ending time

Note 1 to entry: *BoundingValues* are the values that are associated with the starting and ending time of a *ProcessingInterval* specified when reading from the historian. *BoundingValues* may be required by *Clients* to determine the starting and ending values when requesting *raw data* over a time range. If a *raw data* value exists at the start or end point, it is considered the bounding value even though it is part of the data request. If no *raw data* value exists at the start or end point, then the *Server* will determine the boundary value, which may require data from a data point outside of the requested range. See 4.4 for details on using *BoundingValues*.

3.1.3
HistoricalNode
Object, *Variable*, *Property* or *View* in the *AddressSpace* where a *Client* can access historical data or *Events*

Note 1 to entry: A *HistoricalNode* is a term used in this document to represent any *Object*, *Variable, Property* or *View* in the *AddressSpace* for which a *Client* may read and/or update historical data or *Events*. The terms "*HistoricalNode's* history" or "history of a *HistoricalNode*" will refer to the time series data or *Events* stored for this *HistoricalNode*. The term *HistoricalNode* refers to both *HistoricalDataNodes* and *HistoricalEventNodes*.

3.1.4
HistoricalDataNode
Variable or *Property* in the *AddressSpace* where a *Client* can access historical data

Note 1 to entry: A *HistoricalDataNode* represents any Variable or Property in the *AddressSpace* for which a *Client* may read and/or update historical data. "*HistoricalDataNode*'s history" or "history of a *HistoricalDataNode*" refers to the time series data stored for this *HistoricalNode*. Examples of such data are:

- device data (like temperature sensors),
- calculated data,
- status information (open/closed, moving),
- dynamically changing system data (like stock quotes),
- diagnostic data.

The term *HistoricalDataNodes* is used when referencing aspects of the standard that apply to accessing historical data only.

3.1.5
HistoricalEventNode
Object or *View* in the *AddressSpace* for which a *Client* can access historical *Events*

Note 1 to entry: "*HistoricalEventNode*'s history" or "history of a *HistoricalEventNode*" refers to the time series *Events* stored in some historical system. Examples of such data are:

- *Notifications,*
- system *Alarms,*
- operator action *Events,*
- system triggers (such as new orders to be processed).

The term *HistoricalEventNode* is used when referencing aspects of the standard that apply to accessing historical *Events* only.

3.1.6
modified values
HistoricalDataNode's value that has been changed (or manually inserted or deleted) after it was stored in the historian

Note 1 to entry: For some *Servers*, a lab data entry value is not a *modified value*, but if a user corrects a lab value, the original value would be considered a *modified value*, and would be returned during a request for *modified values*. Also manually inserting a value that was missed by a standard collection system may be considered a *modified value*. Unless specified otherwise, all historical *Services* operate on the current, or most recent, value for the specified *HistoricalDataNode* at the specified timestamp. Requests for *modified values* are used to access values that have been superseded, deleted or inserted. It is up to a system to determine what is considered a *modified value*. Whenever a *Server* has modified data available for an entry in the historical collection it shall set the *ExtraData* bit in the *StatusCode.*

3.1.7
raw data
data that is stored within the historian for a *HistoricalDataNode*

Note 1 to entry: The data may be all data collected for the *DataValue* or it may be some subset of the data depending on the historian and the storage rules invoked when the item's values were saved.

3.1.8
StartTime/EndTime
bounds of a history request which define the time domain

Note 1 to entry: For all requests, a value falling at the end time of the time domain is not included in the domain, so that requests made for successive, contiguous time domains will include every value in the historical collection exactly once.

3.1.9
TimeDomain
interval of time covered by a particular request, or response

Note 1 to entry: In general, if the start time is earlier than or the same as the end time, the time domain is considered to begin at the start time and end just before the end time; if the end time is earlier than the start time, the time domain still begins at the start time and ends just before the end time, with time "running backward" for the particular request and response. In both cases, any value which falls exactly at the end time of the *TimeDomain* is not included in the *TimeDomain*. See the examples in 4.4. *BoundingValues* effect the time domain as described in 4.4.

All timestamps which can legally be represented in a *UtcTime DataType* are valid timestamps, and the *Server* may not return an invalid argument result code due to the timestamp being outside of the range for which the *Server* has data. See IEC 62541-3 for a description of the range and granularity of this *DataType*. *Servers* are expected to handle out-of-bounds timestamps gracefully, and return the proper *StatusCodes* to the *Client.*

3.1.10
Structured History Data
structured data stored in a history collection where parts of the structure are used to uniquely identify the data within the data collection

Note 1 to entry: Most historical data applications assume only one current value per timestamp. Therefore the timestamp of the data is considered the unique identifier for that value. Some data or meta data such as *Annotations* may permit multiple values to exist at a single timestamp. In such cases the *Server* would use one or more parameters of the *Structured History Data* entry to uniquely identifiy each element within the history collection. *Annotations* are examples of *Structured History Data*.

3.2 Abbreviations

DA Data Access
HA Historical Access
HDA Historical Data Access
UA Unified Architecture

4 Concepts

4.1 General

This standard defines the handling of historical time series data and historical *Event* data in the OPC Unified Architecture. Included is the specification of the representation of historical data and *Events* in the *AddressSpace.*

4.2 Data architecture

A *Server* supporting Historical Access provides *Clients* with transparent access to different historical data and/or historical *Event* sources (e.g. process historians, event historians, etc.).

The historical data or *Events* may be located in a proprietary data collection, database or a short term buffer within the memory. A *Server* supporting Historical Access will provide

historical data and *Events* for all or a subset of the available *Variables*, *Objects, Properties* or *Views* within the *Server AddressSpace*.

Figure 1 illustrates how the *AddressSpace* of a UA *Server* might consist of a broad range of different historical data and/or historical *Event* sources.

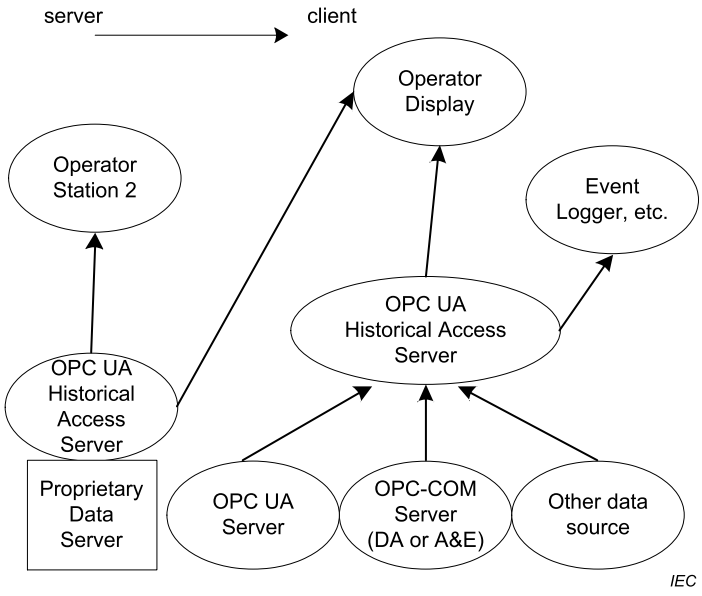


Figure 1 – Possible OPC UA Server supporting Historical Access

The *Server* may be implemented as a standalone OPC UA *Server* that collects data from another OPC UA *Server* or another data source. The *Client* that references the OPC UA *Server* supporting Historical Access for historical data may be simple trending packages that just desire values over a given time frame or they may be complex reports that require data in multiple formats.

4.3 Timestamps

The nature of OPC UA Historical Access requires that a single timestamp reference be used to relate the multiple data points, and the *Client* may request which timestamp will be used as the reference. See IEC 62541-4 for details on the *TimestampsToReturn* enumeration. An OPC UA *Server* supporting Historical Access will treat the various timestamp settings as described below. A HistoryRead with invalid settings will be rejected with Bad_TimestampsToReturnInvalid (see IEC 62541-4).

For *HistoricalDataNodes*, the SourceTimestamp is used to determine which historical data values are to be returned.

The request is in terms of *SourceTimestamp* but the reply could be in *SourceTimestamp*, *ServerTimestamp* or both timestamps. If the reply has the *Server* timestamp the timestamps could fall outside of the range of the requested time.

SOURCE_0 Return the *SourceTimestamp*.

SERVER_1 Return the *ServerTimestamp*.

BOTH_2 Return both the *SourceTimestamp* and *ServerTimestamp*.

NEITHER_3 This is not a valid setting for any HistoryRead accessing *HistoricalDataNodes.*

Any reference to timestamps in this context throughout this standard will represent either *ServerTimestamp* or *SourceTimestamp* as dictated by the type requested in the HistoryRead

Service. Some *Servers* may not support historizing both *SourceTimestamp* and *ServerTimestamp*, but it is expected that all *Servers* will support historizing *SourceTimestamp* (see IEC 62541-7 for details on *Server Profiles*).

If a request is made requesting both *ServerTimestamp* and *SourceTimestamp* and the *Server* is only collecting the *SourceTimestamp* the *Server* shall return Bad_TimestampsToReturnInvalid.

For *HistoricalEventNodes* this parameter does not apply. This parameter is ignored since the entries returned are dictated by the *Event* Filter. See IEC 62541-4 for details.

4.4 Bounding Values and time domain

When accessing *HistoricalDataNodes via* the *HistoryRead Service*, requests can set a flag, returnBounds, indicating that *BoundingValues* are requested. For a complete description of the *Extensible Parameter HistoryReadDetails* that include *StartTime*, *EndTime* and NumValuesPerNode, see 6.4. The concept of Bounding Values and how they affect the time domain that is requested as part of the *HistoryRead* request is further explained in 4.4. 4.4 also provides examples of *TimeDomains* to further illustrate the expected behaviour.

When making a request for historical data using the *HistoryRead Service*, the required parameters include at least 2 of these three parameters: *startTime*, *endTime* and numValuesPerNode. What is returned when Bounding Values are requested varies according to which of these parameters are provided. For a historian that has values stored at 5:00, 5:02, 5:03, 5:05 and 5:06, the data returned when using the Read *Raw* functionality is given by Table 1. In the table, FIRST stands for a tuple with a value of null, a timestamp of the specified *StartTime*, and a *StatusCode* of *Bad_BoundNotFound*. LAST stands for a tuple with a value of null, a timestamp of the specified *EndTime*, and a *StatusCode* of *Bad_BoundNotFound.*

In some cases, attempting to locate bounds, particularly FIRST or LAST points, may be resource intensive for *Servers*. Therefore how far back or forward to look in history for Bounding Values is *Server* dependent, and the *Server* search limits may be reached before a bounding value can be found. There are also cases, such as reading *Annotations* or *Attribute* data where Bounding Values may not be appropriate. For such use cases it is permissible for the *Server* to return a *StatusCode* of *Bad_BoundNotSupported.*

Table 1 – Bounding Value examples

Start Time	End Time	numValuesPerNode	Bounds	Data Returned
5:00	5:05	0	Yes	5:00, 5:02, 5:03, 5:05
5:00	5:05	0	No	5:00, 5:02, 5:03
5:01	5:04	0	Yes	5:00, 5:02, 5:03, 5:05
5:01	5:04	0	No	5:02, 5:03
5:05	5:00	0	Yes	5:05, 5:03, 5:02, 5:00
5:05	5:00	0	No	5:05, 5:03, 5:02
5:04	5:01	0	Yes	5:05, 5:03, 5:02, 5:00
5:04	5:01	0	No	5:03, 5:02
4:59	5:05	0	Yes	FIRST, 5:00, 5:02, 5:03, 5:05
4:59	5:05	0	No	5:00, 5:02, 5:03
5:01	5:07	0	Yes	5:00, 5:02, 5:03, 5:05, 5:06, LAST
5:01	5:07	0	No	5:02, 5:03, 5:05, 5:06
5:00	5:05	3	Yes	5:00, 5:02, 5:03
5:00	5:05	3	No	5:00, 5:02, 5:03
5:01	5:04	3	Yes	5:00, 5:02, 5:03
5:01	5:04	3	No	5:02, 5:03
5:05	5:00	3	Yes	5:05, 5:03, 5:02
5:05	5:00	3	No	5:05, 5:03, 5:02
5:04	5:01	3	Yes	5:05, 5:03, 5:02
5:04	5:01	3	No	5:03, 5:02
4:59	5:05	3	Yes	FIRST, 5:00, 5:02
4:59	5:05	3	No	5:00, 5:02, 5:03
5:01	5:07	3	Yes	5:00, 5:02, 5:03
5:01	5:07	3	No	5:02, 5:03, 5:05
5:00	UNSPECIFIED	3	Yes	5:00, 5:02, 5:03
5:00	UNSPECIFIED	3	No	5:00, 5:02, 5:03
5:00	UNSPECIFIED	6	Yes	5:00, 5:02, 5:03, 5:05, 5:06, LAST[a]
5:00	UNSPECIFIED	6	No	5:00, 5:02, 5:03, 5:05, 5:06
5:07	UNSPECIFIED	6	Yes	5:06, LAST
5:07	UNSPECIFIED	6	No	NODATA
UNSPECIFIED	5:06	3	Yes	5:06,5:05,5:03
UNSPECIFIED	5:06	3	No	5:06,5:05,5:03
UNSPECIFIED	5:06	6	Yes	5:06,5:05,5:03,5:02,5:00,FIRST[b]
UNSPECIFIED	5:06	6	No	5:06, 5:05, 5:03, 5:02, 5:00
UNSPECIFIED	4:48	6	Yes	5:00, FIRST
UNSPECIFIED	4:48	6	No	NODATA
4:48	4:48	0	Yes	FIRST,5:00
4:48	4:48	0	No	NODATA
4:48	4:48	1	Yes	FIRST
4:48	4:48	1	No	NODATA
4:48	4:48	2	Yes	FIRST,5:00
5:00	5:00	0	Yes	5:00,5:02[c]
5:00	5:00	0	No	5:00
5:00	5:00	1	Yes	5:00
5:00	5:00	1	No	5:00
5:01	5:01	0	Yes	5:00, 5:02
5:01	5:01	0	No	NODATA

Start Time	End Time	numValuesPerNode	Bounds	Data Returned
5:01	5:01	1	Yes	5:00
5:01	5:01	1	No	NODATA

a The timestamp of LAST cannot be the specified End Time because there is no specified End Time. In this situation the timestamp for LAST will be equal to the previous timestamp returned plus one second.

b The timestamp of FIRST cannot be the specified End Time because there is no specified Start Time. In this situation the timestamp for FIRST will be equal to the previous timestamp returned minus one second.

c When the Start Time = End Time (there is data at that time), and Bounds is set to True, the start bounds will equal the Start Time and the next data point will be used for the end bounds.

4.5 Changes in AddressSpace over time

Clients use the browse *Services* of the *View Service Set* to navigate through the *AddressSpace* to discover the *HistoricalNodes* and their characteristics. These *Services* provide the most current information about the *AddressSpace*. It is possible and probable that the *AddressSpace* of a *Server* will change over time (i.e. *TypeDefinitions* may change; *NodeIds* may be modified, added or deleted).

Server developers and administrators need to be aware that modifying the *AddressSpace* may impact a *Client's* ability to access historical information. If the history for a *HistoricalNode* is still required, but the *HistoricalNode* is no longer historized, then the *Object* should be maintained in the AddressSpace, with the appropriate *AccessLevel Attribute* and *Historizing Attribute* settings (see IEC 62541-3 for details on access levels).

5 Historical Information Model

5.1 HistoricalNodes

5.1.1 General

The Historical Access model defines additional *Properties* that are applicable for both *HistoricalDataNodes* and *HistoricalEventNodes.*

5.1.2 Annotations Property

The *DataVariable* or *Object* that has *Annotation* data will add the *Annotations Property* (see Table 2 below).

Table 2 – Annotations Property

Name	Use	Data Type	Description
Standard Properties			
Annotations	O	Annotation	The *Annotations Property* is used to indicate that *Annotation* data exists for the history collection exposed by a *HistoricalDataNode*. *Annotation DataType* is defined in 5.5

Since it is not allowed for *Properties* to have *Properties*, the *Annotation Property* is only available for *DataVariables* or *Objects*.

Not every *HistoricalDataNode* in the *AddressSpace* might contain *Annotation* data. The *Annotations Property* indicates whether or not a *HistoricalDataNode* supports *Annotations*. *Annotation* data is accessed using the standard *HistoryRead* functions. *Annotations* are modified, inserted or deleted using the standard *HistoryUpdate* functions.

As with all *HistoricalNodes*, modifications, deletions or additions of *Annotations* will raise the appropriate Historical Audit Event with the corresponding *NodeId*.

5.2 HistoricalDataNodes

5.2.1 General

The Historical Data model defines additional *ObjectTypes and Objects*. These descriptions also include required use cases for *HistoricalDataNodes.*

5.2.2 HistoricalDataConfigurationType

The Historical Access Data model extends the standard type model by defining the *HistoricalDataConfigurationType.* This *Object* defines the general characteristics of a *Node* that defines the historical configuration of any *HistoricalDataNode* that is defined to contain history. It is formally defined in Table 3.

All *Instances* of the *HistoricalDataConfigurationType* use the standard *BrowseName* as defined in Table 6.

Table 3 – HistoricalDataConfigurationType definition

Attribute	**Value**				
BrowseName	HistoricalDataConfigurationType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasComponent	Object	AggregateConfiguration	--	AggregateConfigurationType	Mandatory
HasComponent	Object	AggregateFunctions	--	FolderType	Optional
HasProperty	Variable	Stepped	Boolean	PropertyType	Mandatory
HasProperty	Variable	Definition	String	PropertyType	Optional
HasProperty	Variable	MaxTimeInterval	Duration	PropertyType	Optional
HasProperty	Variable	MinTimeInterval	Duration	PropertyType	Optional
HasProperty	Variable	ExceptionDeviation	Double	PropertyType	Optional
HasProperty	Variable	ExceptionDeviationFormat	Enum	PropertyType	Optional
HasProperty	Variable	StartOfArchive	UtcTime	PropertyType	Optional
HasProperty	Variable	StartOfOnlineArchive	UtcTime	PropertyType	Optional

AggregateConfiguration Object represents the browse entry point for information on how the *Server* treats *Aggregate* specific functionality such as handling *Uncertain data*. This *Object* is required to be present even if it contains no *Aggregate* configuration *Objects*. *Aggregates* are defined in IEC 62541-13.

AggregateFunctions is an entry point to browse to all *Aggregate* capabilities supported by the *Server* for Historical Access. All *HistoryAggregates* supported by the *Server* should be able to be browsed starting from this *Object*. *Aggregates* are defined in IEC 62541-13.

The *Stepped Variable* specifies whether the historical data was collected in such a manner that it should be displayed as *SlopedInterpolation* (sloped line between points) or as *SteppedInterpolation* (vertically-connected horizontal lines between points) when *raw data* is examined. This *Property* also effects how some *Aggregates* are calculated. A value of True indicates the stepped interpolation mode. A value of False indicates *SlopedInterpolation* mode. The default value is False.

The *Definition Variable* is a vendor-specific, human readable string that specifies how the value of this *HistoricalDataNode* is calculated. Definition is non-localized and will often contain an equation that can be parsed by certain *Clients*.

Example: *Definition*::= “(TempA – 25) + TempB”

The *MaxTimeInterval Variable* specifies the maximum interval between data points in the history repository regardless of their value change (see IEC 62541-3 for definition of *Duration*).

The *MinTimeInterval Variable* specifies the minimum interval between data points in the history repository regardless of their value change (see IEC 62541-3 for definition of *Duration*).

The *ExceptionDeviation Variable* specifies the minimum amount that the data for the *HistoricalDataNode* shall change in order for the change to be reported to the history database.

The *ExceptionDeviationFormat Variable* specifies how the ExceptionDeviation is determined. Its values are defined in Table 4.

The *StartOfArchive Variable* specifies the date before which there is no data in the archive either online or offline.

The *StartOfOnlineArchive Variable* specifies the date of the earliest data in the online archive.

Table 4 – ExceptionDeviationFormat Values

Value	Description
ABSOLUTE_VALUE_0	ExceptionDeviation is an absolute Value.
PERCENT_OF_VALUE_1	ExceptionDeviation is a percentage of Value.
PERCENT_OF_RANGE_2	ExceptionDeviation is a percentage of InstrumentRange (see IEC 62541-8).
PERCENT_OF_EU_RANGE_3	ExceptionDeviation is a percentage of EURange (see IEC 62541-8).
UNKNOWN_4	ExceptionDeviation type is Unknown or not specified.

5.2.3 HasHistoricalConfiguration ReferenceType

This *ReferenceType* is a concrete *ReferenceType* that can be used directly. It is a subtype of the *Aggregates ReferenceType* and will be used to refer from a Historical *Node* to one or more *HistoricalDataConfigurationType Objects*.

The semantic indicates that the target *Node* is “used” by the source *Node* of the *Reference*. Figure 2 informally describes the location of this *ReferenceType* in the OPC UA hierarchy. Its representation in the *AddressSpace* is specified in Table 5.

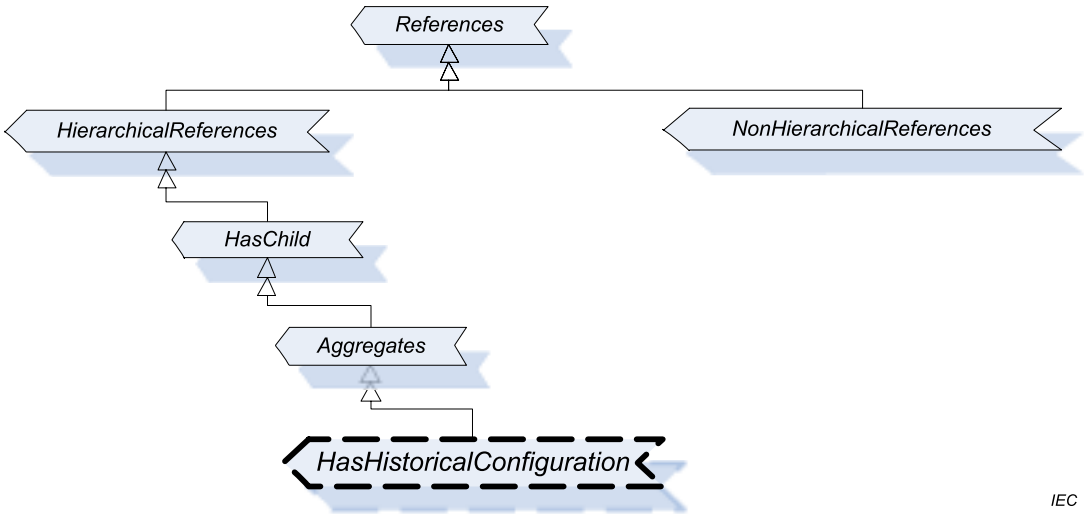


Figure 2 – ReferenceType hierarchy

Table 5 – HasHistoricalConfiguration ReferenceType

Attributes	**Value**		
BrowseName	HasHistoricalConfiguration		
InverseName	HistoricalConfigurationOf		
Symmetric	False		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**Comment**
The subtype of Aggregates *ReferenceType* is defined in IEC 62541-5.			

5.2.4 Historical Data Configuration Object

This *Object* is used as the browse entry point for information about *HistoricalDataNode* configuration. The content of this *Object* is already defined by its type definition in Table 3. It is formally defined in Table 6. If a *HistoricalDataNode* has configuration defined then one instance shall have a *BrowseName* of 'HA Configuration'. Additional configurations may be defined with different *BrowseNames*. All Historical Configuration *Objects* shall be referenced using the *HasHistoricalConfiguration ReferenceType*. It is also highly recommended that display names are chosen that clearly describe the historical configuration e.g. "1 Second Collection", "Long Term Configuration" etc.

Table 6 – Historical Access configuration definition

Attribute	**Value**				
BrowseName	HA Configuration				
References	**Node Class**	**BrowseName**	**DataType**	**TypeDefinition**	**Modelling Rule**
HasTypeDefinition	Object Type	HistoricalDataConfigurationType	Defined in Table 3		

5.2.5 HistoricalDataNodes Address Space Model

HistoricalDataNodes are always a part of other *Nodes* in the *AddressSpace.* They are never defined by themselves. A simple example of a container for *HistoricalDataNodes* would be a "Folder Object".

Figure 3 illustrates the basic *AddressSpace* Model of a *DataVariable* that includes History.

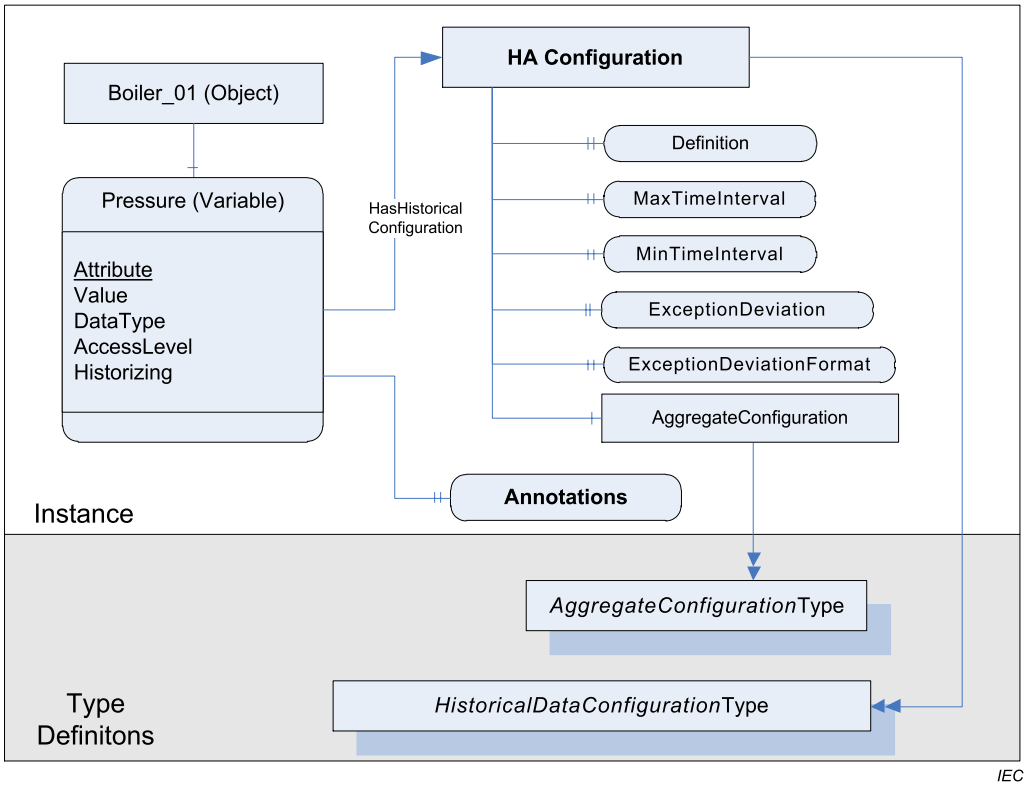


Figure 3 – Historical Variable with Historical Data Configuration and Annotations

Each *HistoricalDataNode* with history shall have the *Historizing Attribute* (see IEC 62541-3) defined and may reference a *HistoricalAccessConfiguration Object*. In the case where the HistoricalDataNode is itself a *Property* then the *HistoricalDataNode* inherits the values from the Parent of the *Property*.

Not every *Variable* in the *AddressSpace* might contain history data. To see if history data is available, a *Client* will look for the HistoryRead/Write states in the *AccessLevel Attribute* (see IEC 62541-3 for details on use of this *Attribute*).

Figure 3 only shows a subset of *Attributes* and *Properties*. Other *Attributes* that are defined for *Variables* in IEC 62541-3, may also be available.

5.2.6 Attributes

Subclause 5.2.6 lists the *Attributes* of *Variables* that have particular importance for historical data. They are specified in detail in IEC 62541-3.

- AccessLevel,
- Historizing.

5.3 HistoricalEventNodes

5.3.1 General

The Historical *Event* model defines additional *Properties*. These descriptions also include required use cases for *HistoricalEventNodes.*

Historical Access of *Events* uses an *EventFilter*. It is important to understand the differences between applying an *EventFilter* to current *Event Notifications*, and historical *Event* retrieval.

In real time monitoring *Events* are received via *Notifications* when subscribing to an *EventNotifier*. The *EventFilter* provides the filtering and content selection of *Event Subscriptions*. If an *Event Notification* conforms to the filter defined by the *where* parameter of the *EventFilter*, then the *Notification* is sent to the *Client*.

In historical *Event* retrieval the *EventFilter* represents the filtering and content selection used to describe what parameters of *Events* are available in history. These may or may not include all of the parameters of the real-time *Event*, i.e. not all fields available when the *Event* was generated may have been stored in history.

The *HistoricalEventFilter* may change over time so a *Client* may specify any field for any *EventType* in the *EventFilter*. If a field is not stored in the historical collection then the field is set to null when it is referenced in the *selectClause* or the *whereClause*.

5.3.2 HistoricalEventFilter Property

A *HistoricalEventNode* that has *Event* history available will provide the *Property*. This *Property* is formally defined in Table 7.

Table 7 – Historical Events Properties

Name	Use	Data Type	Description
Standard Properties			
HistoricalEventFilter	M	EventFilter	A filter used by the *Server* to determine which *HistoricalEventNode* fields are available in history. It may also include a where clause that indicates the types of *Events* or restrictions on the *Events* that are available via the *HistoricalEventNode.* The *HistoricalEventFilter Property* can be used as a guideline for what *Event* fields the Historian is currently storing. But this field may have no bearing on what *Event* fields the Historian is capable of storing.

5.3.3 HistoricalEventNodes Address Space Model

HistoricalEventNodes are *Objects* or *Views* in the *AddressSpace* that expose historical *Events*. These *Nodes* are identified via the *EventNotifier Attribute*, and provide some historical subset of the *Events* generated by the *Server*.

Each *HistoricalEventNode* is represented by an *Object* or View with a specific set of Attributes. The *HistoricalEventFilter Property* specifies the fields available in the history.

Not every *Object* or *View* in the *AddressSpace* may be a *HistoricalEventNode*. To qualify as *HistoricalEventNodes*, a *Node* has to contain historical *Events*. To see if historical *Events* are available, a *Client* will look for the HistoryRead/Write states in the *EventNotifier Attribute*. See IEC 62541-3 for details on the use of this *Attribute.*

Figure 4 illustrates the basic *AddressSpace* Model of an *Event* that includes History.

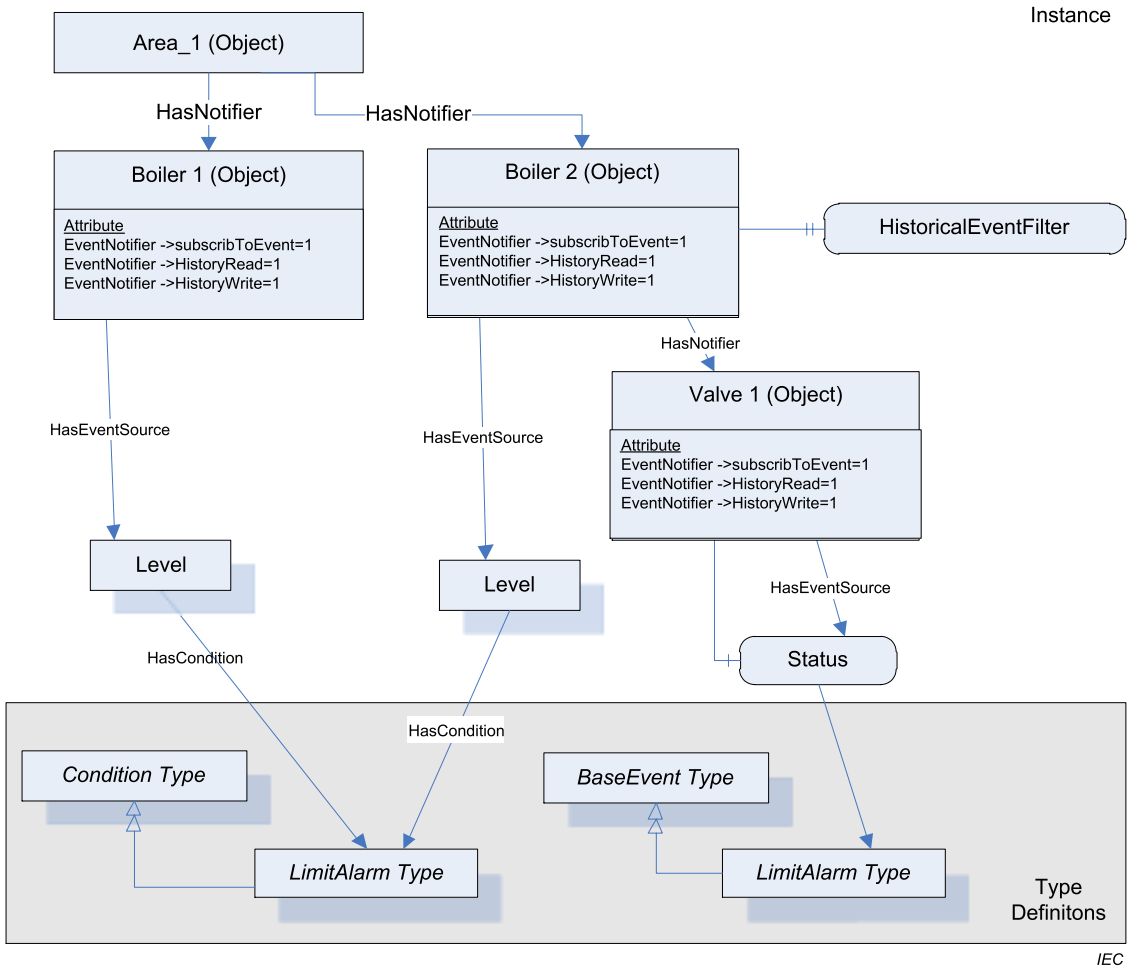


Figure 4 – Representation of an Event with History in the AddressSpace

5.3.4 HistoricalEventNodes Attributes

Subclause 5.3.4 lists the *Attributes* of *Objects* or *Views* that have particular importance for historical *Events*. They are specified in detail in IEC 62541-3. The following *Attributes* are particularly important for *HistoricalEventNodes*.

- EventNotifier

The *EventNotifier Attribute* is used to indicate if the *Node* can be used to read and/or update historical *Events*.

5.4 Exposing supported functions and capabilities

5.4.1 General

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. The *Objects* are outlined in IEC TR 62541-1.

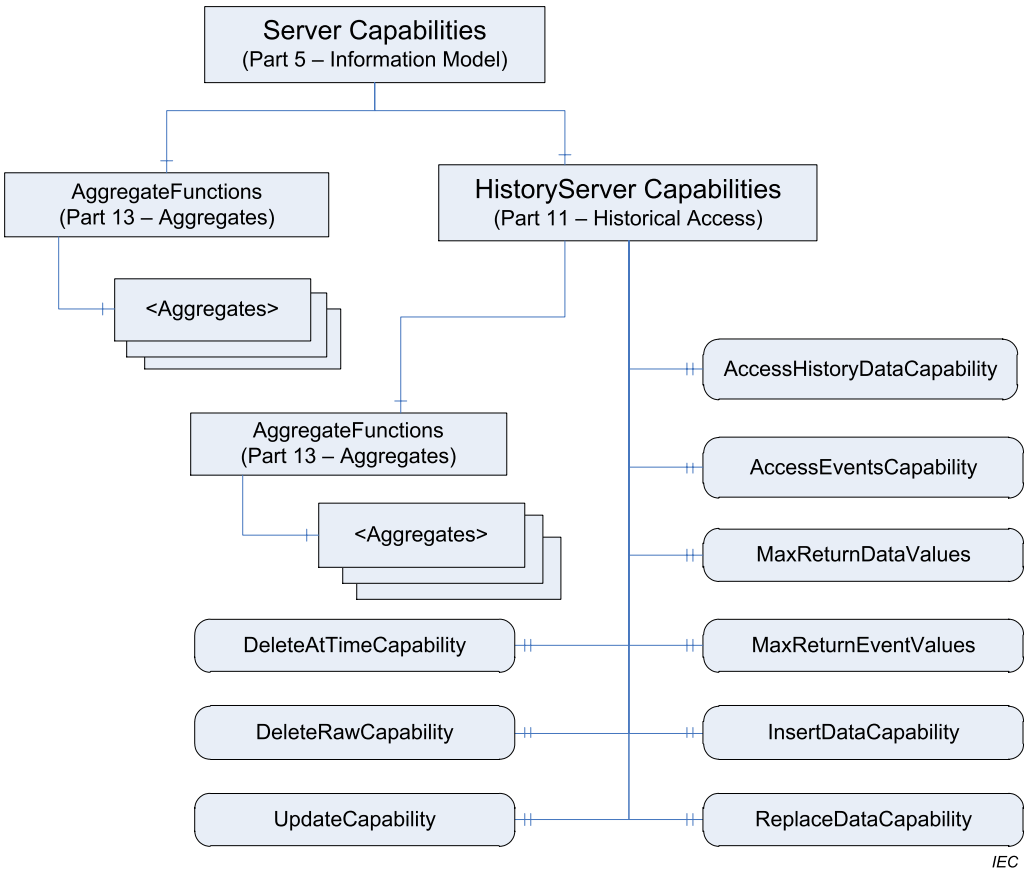


Figure 5 – Server and HistoryServer Capabilities

5.4.2 HistoryServerCapabilitiesType

The *ServerCapabilitiesType Objects* for any OPC UA *Server* supporting Historical Access shall contain a *Reference* to a *HistoryServerCapabilitiesType Object*.

The content of this *BaseObjectType* is already defined by its type definition in IEC 62541-5. The *Object* extensions are formally defined in Table 8.

These properties are intended to inform a *Client* of the general capabilities of the *Server*. They do not guarantee that all capabilities will be available for all *Nodes*. For example not all *Nodes* will support *Events*, or in the case of an aggregating *Server* where underlying *Servers* may not support *Insert* or a particular *Aggregate*. In such cases the *HistoryServerCapabilities Property* would indicate the capability is supported, and the *Server* would return appropriate *StatusCodes* for situations where the capability does not apply.

Table 8 – HistoryServerCapabilitiesType Ddefinition

Attribute	Value				
BrowseName	HistoryServerCapabilitiesType				
IsAbstract	False				
References	**NodeClass**	**Browse Name**	**Data Type**	**Type Definition**	**ModelingRule**
HasProperty	Variable	AccessHistoryDataCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	AccessHistoryEventsCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	MaxReturnDataValues	UInt32	PropertyType	Mandatory
HasProperty	Variable	MaxReturnEventValues	UInt32	PropertyType	Mandatory
HasProperty	Variable	InsertDataCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	ReplaceDataCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	UpdateDataCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	DeleteRawCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	DeleteAtTimeCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	InsertEventCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	ReplaceEventCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	UpdateEventCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	DeleteEventCapability	Boolean	PropertyType	Mandatory
HasProperty	Variable	InsertAnnotationsCapability	Boolean	PropertyType	Mandatory
HasComponent	Object	AggregateFunctions	--	FolderType	Mandatory

All UA *Servers* that support Historical Access shall include the *HistoryServerCapabilities* as part of its *ServerCapabilities*.

The *AccessHistoryDataCapability Variable* defines if the *Server* supports access to historical data values. A value of True indicates the *Server* supports access to the history for *HistoricalNodes*, a value of False indicates the *Server* does not support access to the history for *HistoricalNodes*. The default value is False. At least one of *AccessHistoryDataCapability* or *AccessHistoryEventsCapability* shall have a value of True for the *Server* to be a valid OPC UA *Server* supporting Historical Access.

The *AccessHistoryEventCapability Variable* defines if the server supports access to historical *Events*. A value of True indicates the server supports access to the history of *Events*, a value of False indicates the *Server* does not support access to the history of *Events*. The default value is False. At least one of *AccessHistoryDataCapability* or *AccessHistoryEventsCapability* shall have a value of True for the *Server* to be a valid OPC UA *Server* supporting Historical Access.

The *MaxReturnDataValues Variable* defines the maximum number of values that can be returned by the *Server* for each *HistoricalNode* accessed during a request. A value of 0 indicates that the *Server* forces no limit on the number of values it can return. It is valid for a *Server* to limit the number of returned values and return a continuation point even if *MaxReturnValues* = 0. For example, it is possible that although the *Server* does not impose any restrictions, the underlying system may impose a limit that the *Server* is not aware of. The default value is 0.

Similarily, the *MaxReturnEventValues* specifies the maximum number of *Events* that a *Server* can return for a *HistoricalEventNode*.

The *InsertDataCapability Variable* indicates support for the Insert capability. A value of True indicates the *Server* supports the capability to insert new data values in history, but not overwrite existing values. The default value is False.

The *ReplaceDataCapability Variable* indicates support for the Replace capability. A value of True indicates the *Server* supports the capability to replace existing data values in history, but will not insert new values. The default value is False.

The *UpdateDataCapability Variable* indicates support for the Update capability. A value of True indicates the *Server* supports the capability to insert new data values into history if none exists, and replace values that currently exist. The default value is False.

The *DeleteRawCapability Variable* indicates support for the delete raw values capability. A value of True indicates the *Server* supports the capability to delete *raw data* values in history. The default value is False.

The *DeleteAtTimeCapability Variable* indicates support for the delete at time capability. A value of True indicates the *Server* supports the capability to delete a data value at a specified time. The default value is False.

The *InsertEventCapability Variable* indicates support for the Insert capability. A value of True indicates the *Server* supports the capability to insert new *Events* in history. An insert is not a replace. The default value is False.

The *ReplaceEventCapability Variable* indicates support for the Replace capability. A value of True indicates the *Server* supports the capability to replace existing *Events* in history. A replace is not an insert. The default value is False.

The *UpdateEventCapability Variable* indicates support for the Update capability. A value of True indicates the *Server* supports the capability to insert new *Events* into history if none exists, and replace values that currently exist. The default value is False.

The *DeleteEventCapability Variable* indicates support for the deletion of *Events* capability. A value of True indicates the *Server* supports the capability to delete *Events* in history. The default value is False.

The *InsertAnnotationCapability Variable* indicates support for *Annotations*. A value of True indicates the *Server* supports the capability to insert *Annotations*. Some *Servers* that support Inserting of *Annotations* will also support editing and deleting of *Annotations*. The default value is False.

AggregateFunctions is an entry point to browse to all *Aggregate* capabilities supported by the *Server* for Historical Access. All *HistoryAggregates* supported by the *Server* should be able to be browsed starting from this *Object*. *Aggregates* are defined in IEC 62541-13. If the *Server* does not support *Aggregates* the *Folder* is left empty.

5.5 Annotation DataType

This *DataType* describes *Annotation* information for the history data items. Its elements are defined in Table 9.

Table 9 – Annotation Structure

Name	Type	Description
Annotation	Structure	
message	String	*Annotation* message or text.
username	String	The user that added the *Annotation*, as supplied by the underlying system.
annotationTime	UtcTime	The time the *Annotation* was added. This will probably be different than the *SourceTimestamp.*

5.6 Historical Audit Events

5.6.1 General

AuditEvents are generated as a result of an action taken on the *Server* by a *Client* of the *Server*. For example, in response to a *Client* issuing a write to a *Variable*, the *Server* would generate an *AuditEvent* describing the *Variable* as the source and the user and *Client Session* as the initiators of the *Event*. Not all *Servers* support auditing, but if a *Server* supports auditing then it shall support audit *Events* as described in 5.6. *Profiles* (see IEC 62541-7) can be used to determine if a *Server* supports auditing. *Servers* shall generate *Events* of the *AuditHistoryUpdateEventType* or a sub-type of this type for all invocations of the *HistoryUpdate Service* on any *HistoricalNode*. See IEC 62541-3 and IEC 62541-5 for details on the *AuditHistoryUpdateEventType* model. In the case where the *HistoryUpdate Service* is invoked to insert Historical *Events*, the *AuditHistoryEventUpdateEventType Event* shall include the EventId of the inserted *Event* and a description that indicates that the *Event* was inserted. In the case where the *HistoryUpdate Service* is invoked to delete records, the *AuditHistoryDeleteEventType* or one of its sub-types shall be generated. See 6.7 for details on updating historical data or *Events*.

In particular using the Delete raw or Delete modified functionality shall generate an *AuditHistoryRawModifyDeleteEventType Event* or a sub-type of it. Using the Delete at time functionality shall generate an *AuditHistoryAtTimeDeleteEventType Event* or a sub-type of it. Using the Delete *Event* functionality shall generate an *AuditHistoryEventDeleteEventType Event* or a sub-type of it. All other updates shall follow the guidelines provided in the *AuditHistoryUpdateEventType* model.

5.6.2 AuditHistoryEventUpdateEventType

This is a subtype of AuditHistoryUpdateEventType and is used for categorization of History *Event* update related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 10.

Table 10 – AuditHistoryEventUpdateEventType definition

Attribute	Value				
BrowseName	AuditHistoryEventUpdateEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryUpdateEventType* defined in IEC 62541-3, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Mandatory
HasProperty	Variable	PerformInsertReplace	PerformUpdateType	PropertyType	Mandatory
HasProperty	Variable	Filter	EventFilter	PropertyType	Mandatory
HasProperty	Variable	NewValues	HistoryEventFieldList [ ]	PropertyType	Mandatory
HasProperty	Variable	OldValues	HistoryEventFieldList [ ]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditHistoryUpdateEventType*. Their semantic is defined in IEC 62541-5.

The *UpdateNode* identifies the *Attribute* that was written on the *SourceNode*.

The *PerformInsertReplace* enumeration reflects the parameter on the *Service* call.

The *Filter* reflects the *Event* filter passed on the call to select the *Events* that are to be updated.

The *NewValues* identify the value that was written to the *Event*.

The *OldValues* identify the value that the *Events* contained before the update. It is acceptable for a *Server* that does not have this information to report a null value. In the case of an insert it is expected to be a null value.

Both the *NewValues* and the *OldValues* will contain *Events* with the appropriate fields, each with appropriately encoded values.

5.6.3 AuditHistoryValueUpdateEventType

This is a subtype of *AuditHistoryUpdateEventType* and is used for categorization of history value update related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 11.

Table 11 – AuditHistoryValueUpdateEventType definition

Attribute	**Value**				
BrowseName	AuditHistoryValueUpdateEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryUpdateEventType* defined in IEC 62541-3, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Mandatory
HasProperty	Variable	PerformInsertReplace	PerformUpdateType	PropertyType	Mandatory
HasProperty	Variable	NewValues	DataValue[]	PropertyType	Mandatory
HasProperty	Variable	OldValues	DataValue[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditHistoryUpdateEventType*. Their semantic is defined in IEC 62541-5.

The *UpdatedNode* identifies the *Attribute* that was written on the *SourceNode*.

The *PerformInsertReplace* enumeration reflects the parameter on the *Service* call.

The *NewValues* identify the value that was written to the *Event*.

The *OldValues* identify the value that the *Event* contained before the write. It is acceptable for a *Server* that does not have this information to report a null value. In the case of an insert it is expected to be a null value.

Both the *NewValues* and the *OldValues* will contain a value in the *DataType* and encoding used for writing the value.

5.6.4 AuditHistoryDeleteEventType

This is a subtype of *AuditHistoryUpdateEventType* and is used for categorization of history delete related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 12.

Table 12 – AuditHistoryDeleteEventType definition

Attribute	**Value**				
BrowseName	AuditHistoryDeleteEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryUpdateEventType* defined in IEC 62541-3, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Mandatory
HasSubtype	ObjectType	AuditHistoryRawModifyDeleteEventType			
HasSubtype	ObjectType	AuditHistoryAtTimeDeleteEventType			
HasSubtype	ObjectType	AuditHistoryEventDeleteEventType			

This *EventType* inherits all *Properties* of the *AuditUpdateEventType*. Their semantic is defined in IEC 62541-5.

The *NodeId* identifies the *NodeId* that was used for the delete operation.

5.6.5 AuditHistoryRawModifyDeleteEventType

This is a subtype of *AuditHistoryDeleteEventType* and is used for categorization of history delete related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 13.

Table 13 – AuditHistoryRawModifyDeleteEventType definition

Attribute	**Value**				
BrowseName	AuditHistoryRawModifyDeleteEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryDeleteEventType* defined in Table 12, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	IsDeleteModified	Boolean	PropertyType	Mandatory
HasProperty	Variable	StartTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	EndTime	UtcTime	PropertyType	Mandatory
HasProperty	Variable	OldValues	DataValue[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditHistoryDeleteEventType*. Their semantic is defined in 5.6.4.

The *isDeleteModified* reflects the isDeleteModified parameter of the call.

The *StartTime* reflects the starting time parameter of the call.

The *EndTime* reflects the ending time parameter of the call.

The *OldValues* identify the value that history contained before the delete. A *Server* should report all deleted values. It is acceptable for a *Server* that does not have this information to report a null value. The *OldValues* will contain a value in the *DataType* and encoding used for writing the value.

5.6.6 AuditHistoryAtTimeDeleteEventType

This is a subtype of *AuditHistoryDeleteEventType* and is used for categorization of history delete related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 14.

Table 14 – AuditHistoryAtTimeDeleteEventType definition

Attribute	**Value**				
BrowseName	AuditHistoryAtTimeDeleteEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryDeleteEventType* defined in Table 12, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	ReqTimes	UtcTime[]	PropertyType	Mandatory
HasProperty	Variable	OldValues	DataValues[]	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditHistoryDeleteEventType*. Their semantic is defined in 5.6.7.

The *ReqTimes* reflect the request time parameter of the call.

The *OldValues* identifies the value that history contained before the delete. A *Server* should report all deleted values. It is acceptable for a *Server* that does not have this information to report a null value. The *OldValues* will contain a value in the *DataType* and encoding used for writing the value.

5.6.7 AuditHistoryEventDeleteEventType

This is a subtype of *AuditHistoryDeleteEventType* and is used for categorization of history delete related *Events*. This type follows all the behaviour of its parent type. Its representation in the *AddressSpace* is formally defined in Table 15.

Table 15 – AuditHistoryEventDeleteEventType definition

Attribute	**Value**				
BrowseName	AuditHistoryEventDeleteEventType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *AuditHistoryDeleteEventType* defined in Table 12, i.e. it has *HasProperty References* to the same *Nodes*.					
HasProperty	Variable	EventIds	ByteString[]	PropertyType	Mandatory
HasProperty	Variable	OldValues	HistoryEventFieldList	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditHistoryDeleteEventType*. Their semantic is defined in 5.6.4.

The *EventIds* reflect the EventIds parameter of the call.

The *OldValues* identify the value that history contained before the delete. A *Server* should report all deleted values. It is acceptable for a *Server* that does not have this information to report a null value. The *OldValues* will contain an *Event* with the appropriate fields, each with appropriately encoded values.

6 Historical Access specific usage of Services

6.1 General

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

- The Browse Service Set or Query Service Set to detect *HistoricalNodes* and their configuration.
- The *HistoryRead* and *HistoryUpdate* Services of the Attribute Service Set to read and update history of *HistoricalNodes.*

6.2 Historical Nodes StatusCodes

6.2.1 Overview

6.2 defines additional codes and rules that apply to the *StatusCode* when used for *HistoricalNodes*.

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 historical data and/or *Events*.

6.2.2 Operation level result codes

In OPC UA Historical Access 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 historical data and/or *Events*, additional information beyond the basic quality and call result code needs to be conveyed to the *Client*, for example, whether the value is actually stored in the data repository, whether the result was *Interpolated*, whether all data inputs to a calculation were of good quality, etc.

In the following, Table 16 contains codes with *Bad* severity indicating a failure; Table 17 contains *Good* (success) codes.

It is important to note that these are the codes that are specific for OPC UA Historical Access and supplement the codes that apply to all types of data and are therefore defined in IEC 62541-4, IEC 62541-8 and IEC 62541-13.

Table 16 – Bad operation level result codes

Symbolic Id	Description
Bad_NoData	No data exists for the requested time range or *Event* filter.
Bad_BoundNotFound	No data found to provide upper or lower bound value.
Bad_BoundNotSupported	Bounding Values are not applicable or the *Server* has reached its search limit and will not return a bound.
Bad_DataLost	Data is missing due to collection started/stopped/lost.
Bad_DataUnavailable	Expected data is unavailable for the requested time range due to an un-mounted volume, an off-line historical collection, or similar reason for temporary unavailability.
Bad_EntryExists	The data or *Event* was not successfully inserted because a matching entry exists.
Bad_NoEntryExists	The data or *Event* was not successfully updated because no matching entry exists.
Bad_TimestampNotSupported	The *Client* requested history using a TimestampsToReturn the *Server* does not support (i.e. requested *Server* Timestamp when *Server* only supports SourceTimestamp).
Bad_InvalidArgument	One or more arguments are invalid or missing.
Bad_AggregateListMismatch	The list of Aggregates does not have the same length as the list of operations.
Bad_AggregateConfigurationRejected	The *Server* does not support the specified AggregateConfiguration for the Node.
Bad_AggregateNotSupported	The specified Aggregate is not valid for the specified *Node*.
Bad_ArgumentsMissing	See IEC 62541-4:2015, Table 63, for the description of this result code.
Bad_TypeDefinitionInvalid	See IEC 62541-4:2015, Table 166, for the description of this result code.
Bad_SourceNodeIdInvalid	See IEC 62541-4:2015, Table 166, for the description of this result code.
Bad_OutOfRange	See IEC 62541-4:2015, Table 166, for the description of this result code.
Bad_NotSupported	See IEC 62541-4:2015, Table 166, for the description of this result code.
Bad_IndexRangeInvalid	See IEC 62541-4:2015, Table 166, for the description of this result code.
Bad_NotWriteable	See IEC 62541-4:2015, Table 166, for the description of this result code.

Table 17 – Good operation level result codes

Symbolic Id	Description
Good_NoData	No data exists for the requested time range or *Event* filter.
Good_EntryInserted	The data or *Event* was successfully inserted into the historical database
Good_EntryReplaced	The data or *Event* field was successfully replaced in the historical database
Good_DataIgnored	The *Event* field was ignored and was not inserted into the historical database.

It may be noted that there are both Good and Bad Status codes that deal with cases of no data or missing data. In general *Good_NoData* is used for cases where no data was found when performing a simple 'Read' request. *Bad_NoData i*s used in cases where some action is requested on an interval and no data could be found. The distinction exists if users are attempting an action on a given interval where they would expect data to exist, or would like to be notified that the requested action could not be performed.

Good_NoData is returned for cases such as:

– ReadEvents where *startTime=endTime,*

– ReadEvent data is requested and does not exist,

– ReadRaw where data is requested and does not exist.

Bad_NoData is returned for cases such as:

- ReadEvent data is requested and underlying historian does not support the requested field,
- ReadProcessed where data is requested and does not exist,
- Any Delete requests where data does not exist.

The above use cases are illustrative examples. Detailed explanations on when each status code is returned are found in 6.4 and 6.7

6.2.3 Semantics changed

The *StatusCode* in addition contains an informational bit called *Semantics Changed* (see IEC 62541-4).

UA *Servers* that implement OPC UA Historical Access should not set this bit; rather they should propagate the *StatusCode* which has been stored in the data repository. The *Client* should be aware that the returned data values may have this bit set.

6.3 Continuation Points

The *continuationPoint* parameter in the *HistoryRead Service* is used to mark a point from which to continue the read if not all values could be returned in one response. The value is opaque for the *Client* and is only used to maintain the state information for the *Server* to continue from. For *HistoricalDataNode* requests, a *Server* may use the timestamp of the last returned data item if the timestamp is unique. This can reduce the need in the *Server* to store state information for the continuation point.

The *Client* specifies the maximum number of results per operation in the request *Message*. A *Server* shall not return more than this number of results but it may return fewer results. The *Server* allocates a *ContinuationPoint* if there are more results to return. The *Server* may return fewer results due to buffer issues or other internal constraints. It may also be required to return a *continuationPoint* due to *HistoryRead* parameter constraints. If an *Aggregate* is taking a long time to calculate and is approaching the timeout time, the *Server* may return partial results with a continuation point. This may be done if the calculation is going to take more time than the *Client* timeout. In some cases it may take longer than the *Client* timeout to calculate even one *Aggregate* result. Then the *Server* may return zero results with a continuation point that allows the *Server* to resume the calculation on the next *Client* read call. For additional discussions regarding *ContinuationPoints* and *HistoryRead* please see the individual extensible *HistoryReadDetails* parameter in 6.4

If the *Client* specifies a *ContinuationPoint*, then the *HistoryReadDetails* parameter and the TimestampsToReturn parameter are ignored, because it does not make sense to request different parameters when continuing from a previous call. It is permissible to change the *dataEncoding* parameter with each request.

If the *Client* specifies a *ContinuationPoint* that does not correspond with the last returned *ContinuationPoint* from the *Server*, then the *Server* shall return a *Bad_ContinuationPointInvalid* error.

If the *releaseContinuationPoints* parameter is set in the request the *Server* shall not return any data and shall release all *ContinuationPoints* passed in the request. If the *ContinuationPoint* for an operation is missing or invalid then the *StatusCode* for the operation shall be *Bad_ContinuationPointInvalid.*

6.4 HistoryReadDetails parameters

6.4.1 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 and the details that are specific to that function. See IEC 62541-4 for the definition of *Extensible Parameter*. Table 18 lists the symbolic names of the valid *Extensible Parameter* structures. Some structures will perform different functions based on the setting of its associated parameters. For simplicity a functionality of each structure is listed. For example, text such as 'using the Read modified functionality' refers to the function the *HistoryRead Service* performs using the *Extensible Parameter* structure ReadRawModifiedDetails with the isReadModified Boolean parameter set to TRUE.

Table 18 – HistoryReadDetails parameterTypeIds

Symbolic Name	Functionality	Description
ReadEventDetails	Read event	This structure selects a set of *Events* from the history database by specifying a filter and a time domain for one or more *Objects* or *Views*. See 6.4.2.1. When this parameter is specified the *Server* returns a *HistoryEvent* structure for each operation (see 6.5.4).
ReadRawModifiedDetails	Read raw	This structure selects a set of values from the history database by specifying a time domain for one or more *Variables*. See 6.4.3.1. When this parameter is specified the *Server* returns a *HistoryData* structure for each operation (see 6.5.2).
ReadRawModifiedDetails	Read modified	This parameter selects a set of *modified values* from the history database by specifying a time domain for one or more *Variables.* See 6.4.3.1. When this parameter is specified the *Server* returns a *HistoryModifiedData* structure for each operation (see 6.5.3).
ReadProcessedDetails	Read processed	This structure selects a set of *Aggregate* values from the history database by specifying a time domain for one or more *Variables*. See 6.4.4.1. When this parameter is specified the *Server* returns a *HistoryData* structure for operation (see 6.5.2)
ReadAtTimeDetails	Read at time	This structure selects a set of raw or interpolated values from the history database by specifying a series of timestamps for one or more *Variables*. See 6.4.5.1. When this parameter is specified the *Server* returns a *HistoryData* structure for each operation (see 6.5.2).

6.4.2 ReadEventDetails structure

6.4.2.1 ReadEventDetails structure details

Table 19 defines the *ReadEventDetails* structure. This parameter is only valid for *Objects* that have the *EventNotifier Attribute* set to TRUE (see IEC 62541-3). Two of the three parameters, *numValuesPerNode*, *startTime*, and *endTime* shall be specified.

Table 19 – ReadEventDetails

Name	Type	Description
ReadEventDetails	Structure	Specifies the details used to perform an *Event* history read.
numValuesPerNode	Counter	The maximum number of values returned for any *Node* over the time range. If only one time is specified, the time range shall extend to return this number of values. The default value of 0 indicates that there is no maximum.
startTime	UtcTime	Beginning of period to read. The default value of *DateTime.MinValue* indicates that the *startTime* is Unspecified.
endTime	UtcTime	End of period to read. The default value of *DateTime.MinValue* indicates that the endTime is Unspecified.
Filter	EventFilter	A filter used by the *Server* to determine which HistoricalEventNode should be included. This parameter shall be specified and at least one *EventField* is required. The *EventFilter* parameter type is an *Extensible parameter* type. It is defined and used in the same manner as defined for monitored data items which are specified in IEC 62541-4. This filter also specifies the *EventFields* that are to be returned as part of the request.

6.4.2.2 Read Event functionality

The *ReadEventDetails* structure is used to read the *Events* from the history database for the specified time domain for one or more *HistoricalEventNodes*. The *Events* are filtered based on the filter structure provided. This filter includes the *EventFields* that are to be returned. For a complete description of filter refer to IEC 62541-4.

The *startTime* and *endTime* are used to filter on the Time field for *Events*.

The time domain of the request is defined by *startTime*, *endTime*, and numValuesPerNode; at least two of these shall be specified. If *endTime* is less than *startTime*, or *endTime* and *numValuesPerNode* alone are specified then the data will be returned in reverse order with later/newer data provided first as if time were flowing backward. If all three are specified then the call shall return up to numValuesPerNode results going from *startTime* to *endTime*, in either ascending or descending order depending on the relative values of *startTime* and *endTime*. If numValuesPerNode is 0 then all of the values in the range are returned. The default value is used to indicate when *startTime*, *endTime* or *numValuesPerNode* are not specified.

It is specifically allowed for the *startTime* and the *endTime* to be identical. This allows the *Client* to request the *Event* at a single instance in time. When the *startTime* and *endTime* are identical then time is presumed to be flowing forward. If no data exists at the time specified then the *Server* shall return the *Good_NoData StatusCode*.

If a *startTime*, *endTime* and *numValuesPerNode* are all provided, and if more than *numValuesPerNode Events* exist within that time range for a given *Node*, then only *numValuesPerNode Events* per *Node* are returned along with a *ContinuationPoint*. When a *ContinuationPoint* is returned, a *Client* wanting the next *numValuesPerNode* values should call *HistoryRead* again with the *continuationPoin*t set.

For an interval in which no data exists, the corresponding *StatusCode* shall be *Good_NoData*.

The filter parameter is used to determine which historical *Events* and their corresponding fields are returned. It is possible that the fields of an *EventType* are available for real time updating, but not available from the historian. In this case a *StatusCode* value will be returned for any *Event* field that cannot be returned. The value of the *StatusCode* shall be *Bad_NoData*.

If the requested *TimestampsToReturn* is not supported for a *Node* then the operation shall return the *Bad_TimestampNotSupported StatusCode*. When reading *Events* this only applies to *Event* fields that are of type *DataValue*.

6.4.3 ReadRawModifiedDetails structure

6.4.3.1 ReadRawModifiedDetails structure details

Table 20 defines the *ReadRawModifiedDetails* structure. Two of the three parameters, *numValuesPerNode*, *startTime*, and *endTime* shall be specified.

Table 20 – ReadRawModifiedDetails

Name	Type	Description
ReadRawModifiedDetails	Structure	Specifies the details used to perform a “raw” or “modified” history read.
isReadModified	Boolean	TRUE for Read Modified functionality, FALSE for Read Raw functionality. Default value is FALSE.
startTime	UtcTime	Beginning of period to read. Set to default value of *DateTime.MinValue* if no specific start time is specified.
endTime	UtcTime	End of period to read. Set to default value of *DateTime.MinValue* if no specific end time is specified.
numValuesPerNode	Counter	The maximum number of values returned for any *Node* over the time range. If only one time is specified, the time range shall extend to return this number of values. The default value 0 indicates that there is no maximum.
returnBounds	Boolean	A Boolean parameter with the following values: TRUE Bounding Values should be returned FALSE All other cases

6.4.3.2 Read raw functionality

When this structure is used for reading *Raw* Values (isReadModified is set to FALSE), it reads the values, qualities, and timestamps from the history database for the specified time domain for one or more *HistoricalDataNodes*. This parameter is intended for use by a *Client* that wants the actual data saved within the historian. The actual data may be compressed or may be all raw data collected for the item depending on the historian and the storage rules invoked when the item values were saved. When returnBounds is TRUE, the Bounding Values for the time domain are returned. The optional Bounding Values are provided to allow the *Client* to interpolate values for the start and end times when trending the actual data on a display.

The time domain of the request is defined by *startTime*, *endTime*, and *numValuesPerNode*; at least two of these shall be specified. If endTime is less than *startTime*, or *endTime* and *numValuesPerNode* alone are specified then the data will be returned in reverse order, with later data coming first as if time were flowing backward. If all three are specified then the call shall return up to *numValuesPerNode* results going from *startTime* to *endTime*, in either ascending or descending order depending on the relative values of *startTime* and *endTime*. If *numValuesPerNode* is 0, then all the values in the range are returned. A default value of *DateTime.MinValue* (see IEC 62541-6) is used to indicate when *startTime* or *endTime* is not specified.

It is specifically allowed for the *startTime* and the *endTime* to be identical. This allows the *Client* to request just one value. When the *startTime* and *endTime* are identical then time is presumed to be flowing forward. It is specifically not allowed for the *Server* to return a *Bad_InvalidArgument StatusCode* if the requested time domain is outside of the *Server*'s range. Such a case shall be treated as an interval in which no data exists.

If a *startTime*, *endTime* and *numValuesPerNode* are all provided and if more than *numValuesPerNode* values exist within that time range for a given *Node* then only

numValuesPerNode values per *Node* are returned along with a *continuationPoint*. When *a continuationPoint* is returned, a *Client* wanting the next numValuesPerNode values should call ReadRaw again with the *continuationPoint* set.

If Bounding Values are requested and a non-zero *numValuesPerNode* was specified then any Bounding Values returned are included in the *numValuesPerNode* count. If *numValuesPerNode* is 1 then only the start bound is returned (the end bound if the reverse order is needed). If *numValuesPerNode* is 2 then the start bound and the first data point are returned (the end bound if reverse order is needed). When Bounding Values are requested and no bounding value is found then the corresponding *StatusCode* entry will be set to *Bad_BoundNotFound*, a timestamp equal to the start or end time as appropriate, and a value of null. How far back or forward to look in history for Bounding Values is *Server* dependent.

For an interval in which no data exists, if Bounding Values are not requested, then the corresponding *StatusCode* shall be *Good_NoData*. If Bounding Values are requested and one or both exist, then the result code returned is Success and the bounding value(s) are returned.

For cases where there are multiple values for a given timestamp, all but the most recent are considered to be *Modified values* and the *Server* shall return the most recent value. If the *Server* returns a value which hides other values at a timestamp then it shall set the *ExtraData* bit in the *StatusCode* associated with that value. If the *Server* contains additional information regarding a value then the ExtraData bit shall also be set. It indicates that ModifiedValues are available for retrieval, see 6.4.3.3.

If the requested TimestampsToReturn is not supported for a *Node*, the operation shall return the *Bad_TimestampNotSupported StatusCode*.

6.4.3.3 Read modified functionality

When this structure is used for reading *Modified* Values (isReadModified is set to TRUE), it reads the values, *StatusCodes*, timestamps, modification type, the user identifier, and the timestamp of the modification from the history database for the specified time domain for one or more *HistoricalDataNodes*. If there are multiple replaced values the *Server* shall return all of them. The updateType specifies what value is returned in the modification record. If the updateType is INSERT the value is the new value that was inserted. If the updateType is anything else the value is the old value that was changed.

The purpose of this function is to read values from history that have been *Modified.* The returnBounds parameter shall be set to FALSE for this case, otherwise the *Server* returns a *Bad_InvalidArgument StatusCode*.

The domain of the request is defined by *startTime*, *endTime*, and *numValuesPerNode*; at least two of these shall be specified. If endTime is less than *startTime*, or *endTime* and *numValuesPerNode* alone are specified, then the data shall be returned in reverse order with the later data coming first. If all three are specified then the call shall return up to numValuesPerNode results going from StartTime to *EndTime*, in either ascending or descending order depending on the relative values of *StartTime* and *EndTime*. If more than *numValuesPerNode* values exist within that time range for a given *Node* then only *numValuesPerNode* values per *Node* are returned along with a *continuationPoint*. When *a continuationPoint* is returned, a *Client* wanting the next *numValuesPerNode* values should call *ReadRaw* again with the *continuationPoint* set. If *numValuesPerNode* is 0 then all of the values in the range are returned. If the *Server* cannot return all *modified values* for a given timestamp in a single response then it shall return modified values with the same timestamp in subsequent calls.

If a value has been modified multiple times then all values for the time are returned. This means that a timestamp can appear in the array more than once. The order of the returned values with the same timestamp should be from the most recent to oldest modification

timestamp, if *startTime* is less than or equal to *endTime*. If *endTime* is less than *startTime*, then the order of the returned values will be from the oldest modification timestamp to the most recent. It is *Server* dependent whether multiple modifications are kept or only the most recent.

A *Server* does not have to create a modification record for data when it is first added to the historical collection. If it does then it shall set the ExtraData bit and the *Client* can read the modification record using a ReadModified call. If the data is subsequently modified the *Server* shall create a second modification record which is returned along with the original modification record whenever a *Client* uses the ReadModified call if the *Server* supports multiple modification records per timestamp.

If the requested TimestampsToReturn is not supported for a *Node* then the operation shall return the *Bad_TimestampNotSupported StatusCode*.

6.4.4 ReadProcessedDetails structure

6.4.4.1 ReadProcessedDetails structure details

Table 21 defines the structure of the ReadProcessedDetails structure.

Table 21 – ReadProcessedDetails

Name	Type	Description
ReadProcessedDetails	Structure	Specifies the details used to perform a “processed” history read.
startTime	UtcTime	Beginning of period to read.
endTime	UtcTime	End of period to read.
ProcessingInterval	Duration	Interval between returned *Aggregate* values. The value 0 indicates that there is no *ProcessingInterval* defined.
aggregateType[]	NodeId	The *NodeId* of the HistoryAggregate object that indicates the list of *Aggregates* to be used when retrieving the processed history. See IEC 62541-13 for details.
aggregateConfiguration	Aggregate Configuration	*Aggregate* configuration structure.
useSeverCapabilitiesDefaults	Boolean	As described in IEC 62541-4.
TreatUncertainAsBad	Boolean	As described in IEC 62541-13.
PercentDataBad	UInt8	As described in IEC 62541-13.
PercentDataGood	UInt8	As described in IEC 62541-13.
UseSlopedExtrapolation	Boolean	As described in IEC 62541-13.

See IEC 62541-13 for details on possible *NodeId* values for the *HistoryAggregateType* parameter.

6.4.4.2 Read processed functionality

This structure is used to compute *Aggregate* values, qualities, and timestamps from data in the history database for the specified time domain for one or more *HistoricalDataNodes*. The time domain is divided into intervals of duration *ProcessingInterval*. The specified *Aggregate* Type is calculated for each interval beginning with *startTime* by using the data within the next *ProcessingInterval*.

For example, this function can provide hourly statistics such as Maximum, Minimum, and Average for each item during the specified time domain when *ProcessingInterval* is 1 h.

The domain of the request is defined by *startTime*, *endTime*, and *ProcessingInterval*. All three shall be specified. If *endTime* is less than *startTime* then the data shall be returned in reverse order with the later data coming first. If *startTime* and *endTime* are the same then the *Server* shall return *Bad_InvalidArgument* as there is no meaningful way to interpret such a case.

The aggregateType[] parameter allows a *Client* to request multiple *Aggregate* calculations per requested NodeId. If multiple *Aggregates* are requested then a corresponding number of entries are required in the *NodesToRead* array.

For example, to request Min *Aggregate* for *NodeId* FIC101, FIC102, and both Min and Max *Aggregates* for *NodeId* FIC103 would require *NodeId* FIC103 to appear twice in the NodesToRead array request parameter.

aggregateType[]	NodesToRead[]
Min	FIC101
Min	FIC102
Min	FIC103
Max	FIC103

If the array of *Aggregates* does not match the array of *NodesToRead* then the *Server* shall return a *StatusCode* of *Bad_AggregateListMismatch.*

The aggregateConfiguration parameter allows a *Client* to override the *Aggregate* configuration settings supplied by the AggregateConfiguration *Object* on a per call basis. See IEC 62541-13 for more information on *Aggregate* configurations. If the *Server* does not support the ability to override the *Aggregate* configuration settings then it shall return a *StatusCode* of *Bad_ AggregateConfigurationRejected.* If the *Aggregate* is not valid for the *Node* then the *StatusCode* shall be *Bad_AggregateNotSupported.*

The values used in computing the *Aggregate* for each interval shall include any value that falls exactly on the timestamp at the beginning of the interval, but shall not include any value that falls directly on the timestamp ending the interval. Thus, each value shall be included only once in the calculation. If the time domain is in reverse order then we consider the later timestamp to be the one beginning the subinterval, and the earlier timestamp to be the one ending it. Note that this means that simply swapping the start and end times will not result in getting the same values back in reverse order as the intervals being requested in the two cases are not the same.

If an *Aggregate* is taking a long time to calculate then the *Server* can return partial results with a continuation point. This might be done if the calculation is going to take more time than the *Client* timeout hint. In some cases it may take longer than the *Client* timeout hint to calculate even one *Aggregate* result. Then the *Server* may return zero results with a continuation point that allows the *Server* to resume the calculation on the next *Client* read call.

Refer to IEC 62541-13 for handling of *Aggregate* specific cases.

6.4.5 ReadAtTimeDetails structure

6.4.5.1 ReadAtTimeDetails structure details

Table 22 defines the ReadAtTimeDetails structure.

Table 22 – ReadAtTimeDetails

Name	Type	Description
ReadAtTimeDetails	Structure	Specifies the details used to perform an “at time” history read.
reqTimes []	UtcTime	The entries define the specific timestamps for which values are to be read.
useSimpleBounds	Boolean	Use SimpleBounds to determine the value at the specific timestamp.

6.4.5.2 Read at time functionality

The ReadAtTimeDetails structure reads the values and qualities from the history database for the specified timestamps for one or more *HistoricalDataNodes*. This function is intended to provide values to correlate with other values with a known timestamp. For example, a *Client* may need to read the values of sensors when lab samples were collected.

The order of the values and qualities returned shall match the order of the timestamps supplied in the request.

When no value exists for a specified timestamp, a value shall be *Interpolated* from the surrounding values to represent the value at the specified timestamp. The interpolation will follow the same rules as the standard *Interpolated Aggregate* as outlined in IEC 62541-13.

If the useSimpleBounds flag is True and Interpolation is required then *SimpleBounds* will be used to calculate the data value.

If a value is found for the specified timestamp, then the *Server* will set the *StatusCode InfoBits* to be *Raw*. If the value is *Interpolated* from the surrounding values, then the *Server* will set the *StatusCode InfoBits* to be *Interpolated.*

If the requested TimestampsToReturnis not supported for a *Node*, then the operation shall return the *Bad_TimestampNotSupported StatusCode*.

6.5 HistoryData parameters returned

6.5.1 Overview

The *HistoryRead Service* returns different types of data depending on whether the request asked for the value *Attribute* of a *Node* or the history *Events* of a *Node*. The historyData is an *Extensible Parameter* whose structure depends on the functions to perform for the *HistoryReadDetails* parameter. See IEC 62541-4 for details on *Extensible Parameters*.

6.5.2 HistoryData type

Table 23 defines the structure of the *HistoryData* used for the data to return in a *HistoryRead*.

Table 23 – HistoryData Details

Name	Type	Description
dataValues[]	DataValue	An array of values of history data for the *Node*. The size of the array depends on the requested data parameters.

6.5.3 HistoryModifiedData type

Table 24 defines the structure of the *HistoryModifiedData* used for the data to return in a *HistoryRead* when IsReadModified = True.

Table 24 – HistoryModifiedData Details

Name	Type	Description
dataValues[]	DataValue	An array of values of history data for the *Node*. The size of the array depends on the requested data parameters.
modificationInfos[]	ModificationInfo	
Username	String	The name of the user that made the modification. Support for this field is optional. A null shall be returned if it is not defined.
modificationTime	UtcTime	The time the modification was made. Support for this field is optional. A null shall be returned if it is not defined.
updateType	HistoryUpdateType	The modification type for the item.

6.5.4 HistoryEvent type

Table 25 defines the HistoryEvent parameter used for Historical *Event* reads.

The HistoryEvent defines a table structure that is used to return *Event* fields to a *Historical Read*. The structure is in the form of a table consisting of one or more *Events*, each containing an array of one or more fields. The selection and order of the fields returned for each *Event* are identical to the selected parameter of the *EventFilter*.

Table 25 – HistoryEvent Details

Name	Type	Description
Events []	HistoryEventFieldList	The list of *Events* being delivered.
eventFields []	BaseDataType	List of selected *Event* fields. This will be a one-to-one match with the fields selected in the *EventFilter*.

6.6 HistoryUpdateType Enumeration

Table 26 defines the HistoryUpdate enumeration.

Table 26 – HistoryUpdateType Enumeration

Name	Description
INSERT_1	Data was inserted.
REPLACE_2	Data was replaced.
UPDATE_3	Data was inserted or replaced.
DELETE_4	Data was deleted.

6.7 PerformUpdateType Enumeration

Table 27 defines the PerformUpdateType enumeration.

Table 27 – PerformUpdateType Enumeration

Name	Description
INSERT_1	Data was inserted.
REPLACE_2	Data was replaced.
UPDATE_3	Data was inserted or replaced.
DELETE_4	Data was deleted.

6.8 HistoryUpdateDetails parameter

6.8.1 Overview

The *HistoryUpdate Service* defined in IEC 62541-4 can perform several different functions. The *historyUpdateDetails* parameter is an *Extensible Parameter* that specifies which function to perform and the details that are specific to that function. See IEC 62541-4 for the definition of *Extensible Parameter*. Table 28 lists the symbolic names of the valid *Extensible Parameter* structures. Some structures will perform different functions based on the setting of its associated parameters. For simplicity a functionality of each structure is listed. For example text such as 'using the Replace data functionality' refers to the function the *HistoryUpdate Service* performs using the *Extensible Parameter* structure *UpdateDataDetails* with the performInsertReplace enumeration parameter set to REPLACE_2.

Table 28 – HistoryUpdateDetails parameter TypeIds

Symbolic Name	Functionality	Description
UpdateDataDetails	Insert data	This function inserts new values into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateDataDetails	Replace data	This function replaces existing values into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateDataDetails	Update data	This function inserts or replaces values into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateStructureDataDetails	Insert data	This function inserts new *Structured History Data* or *Annotations* into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateStructureDataDetails	Replace data	This function replaces existing *Structured History Data* or *Annotations* into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateStructureDataDetails	Update data	This function inserts or replaces *Structured History Data* or *Annotations* into the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateStructureDataDetails	Remove data	This function removes *Structured History Data* or *Annotations* from the history database at the specified timestamps for one or more *HistoricalDataNodes*. The *Variable*'s value is represented by a composite value defined by the *DataValue* data type.
UpdateEventDetails	Insert events	This function inserts new *Events* into the history database for one or more *HistoricalEventNodes*.
UpdateEventDetails	Replace events	This function replaces values of fields in existing *Events* into the history database for one or more *HistoricalEventNodes*.
UpdateEventDetails	Update events	This function inserts new *Events* or replaces existing *Events* in the history database for one or more *HistoricalEventNodes*.
DeleteRawModifiedDetails	Delete raw	This function deletes all values from the history database for the specified time domain for one or more *HistoricalDataNodes*.

Symbolic Name	Functionality	Description
DeleteRawModifiedDetails	Delete modified	Some historians may store multiple values at the same Timestamp. This function will delete specified values and qualities for the specified timestamp for one or more *HistoricalDataNodes*.
DeleteAtTimeDetails	Delete at time	This function deletes all values in the history database for the specified timestamps for one or more *HistoricalDataNodes*.
DeleteEventDetails	Delete event	This function deletes *Events* from the history database for the specified filter for one or more *HistoricalEventNodes*.

The HistoryUpdate *Service* is used to update or delete both *DataValues* and *Events*. For simplicity the term “entry” will be used to mean either *DataValue* or *Event* depending on the context in which it is used. Auditing requirements for History *Services* are described in IEC 62541-4. This description assumes the user issuing the request and the *Server* that is processing the request support the capability to update entries. See IEC 62541-3 for a description of *Attributes* that expose the support of Historical Updates.

6.8.2 UpdateDataDetails structure

6.8.2.1 UpdateDataDetails structure details

Table 29 defines the UpdateDataDetails structure.

Table 29 – UpdateDataDetails

Name	Type	Description
UpdateDataDetails	Structure	The details for insert, replace, and insert/replace history updates.
nodeId	NodeId	Node id of the *Object* to be updated.
performInsertReplace	PerformUpdateType	Value determines which action of insert, replace, or update is performed. **Value** / **Description** INSERT_1 / See 6.8.2.2. REPLACE_2 / See 6.8.2.3. UPDATE_3 / See 6.8.2.4.
updateValues[]	DataValue	New values to be inserted or to replace.

6.8.2.2 Insert data functionality

Setting performInsertReplace = INSERT_1 inserts entries into the history database at the specified timestamps for one or more *HistoricalDataNodes*. If an entry exists at the specified timestamp, then the new entry shall not be inserted; instead the *StatusCode* shall indicate *Bad_EntryExists*.

This function is intended to insert new entries at the specified timestamps, e.g., the insertion of lab data to reflect the time of data collection.

6.8.2.3 Replace data functionality

Setting performInsertReplace = REPLACE_2 replaces entries in the history database at the specified timestamps for one or more *HistoricalDataNodes*. If no entry exists at the specified timestamp, then the new entry shall not be inserted; otherwise the *StatusCode* shall indicate *Bad_NoEntryExists*.

This function is intended to replace existing entries at the specified timestamp, e.g., correct lab data that was improperly processed, but inserted into the history database.

6.8.2.4 Update data functionality

Setting performInsertReplace = UPDATE_3 inserts or replaces entries in the history database for the specified timestamps for one or more *HistoricalDataNodes*. If the item has an entry at the specified timestamp, then the new entry will replace the old one. If there is no entry at that timestamp, then the function will insert the new data.

A *Server* can create a *modified value* for a value being replaced or inserted (see 3.1.6) however it is not required.

This function is intended to unconditionally insert/replace values and qualities, e.g., correction of values for bad sensors.

Good as a *StatusCode* for an individual entry is allowed when the *Server* is unable to say whether there was already a value at that timestamp. If the *Server* can determine whether the new entry replaces an entry that was already there, then it should use *Good_EntryInserted* or *Good_EntryReplaced* to return that information.

6.8.3 UpdateStructureDataDetails structure

6.8.3.1 UpdateStructureDataDetails structure details

Table 30 defines the UpdateStructureDataDetails structure.

Table 30 – UpdateStructureDataDetails

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>UpdateStructureDataDetails</td><td>Structure</td><td>The details for data history updates.</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>Node id of the Object to be updated.</td></tr>
<tr><td>performInsertReplace</td><td>PerformUpdateType</td><td>Value determines which action of insert, replace, or update is performed.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>INSERT_1</td><td>See 6.8.3.3.</td></tr><tr><td>REPLACE_2</td><td>See 6.8.3.4.</td></tr><tr><td>UPDATE_3</td><td>See 6.8.3.5.</td></tr><tr><td>REMOVE_4</td><td>See 6.8.3.6.</td></tr></table></td></tr>
<tr><td>updateValue[]</td><td>DataValue</td><td>New values to be inserted, replaced or removed.</td></tr>
</table>

6.8.3.2 Specfied Uniqueness of Structured History Data

S*tructured History Data* provides metadata describing an entry in the history database. The *Server* shall define what uniqueness means for each *Structured History Data* structure type. For example, a *Server* may only allow one *Annotation* per timestamp which means the timestamp is the unique key for the structure. Another *Server* may allow for multiple *Annotations* to exist per user, so a combination of a username, timestamp, and message may be used as the unique key for the structure. In 6.8.3.3, 6.8.3.4, 6.8.3.5, and 6.8.3.6 the terms '*Structured History Data* exists' and 'at the specified parameters' means a matching entry has been found at the specified timestamp using the *Server's* criteria for uniqueness.

In the case where the Client wishes to Replace a parameter that is part of the uniqueness criteria, then the resulting StatusCode would be *Bad_NoEntryExists*. The Client shall Remove the existing structure and then Insert the new structure.

6.8.3.3 Insert functionality

Setting performInsertReplace = INSERT_1 inserts *Structured History Data* such as *Annotations* into the history database at the specified parameters for one or more *Properties* of *HistoricalDataNodes.*

If a *Structured History Data* entry already exists at the specified parameters the *StatusCode* shall indicate *Bad_EntryExists*.

6.8.3.4 Replace functionality

Setting performInsertReplace = REPLACE_2 replaces *Structured History Data* such as *Annotations* in the history database at the specified parameters for one or more *Properties* of *HistoricalDataNodes.*

If a *Structured History Data* entry does not already exist at the specified parameters, then the *StatusCode* shall indicate *Bad_NoEntryExists*.

6.8.3.5 Update functionality

Setting performInsertReplace = UPDATE_3 inserts or replaces *Structured History Data* such as *Annotations* in the history database at the specified parameters for one or more *Properties* of *HistoricalDataNodes.*

If a *Structure History Data* entry already exists at the specified parameters then it is deleted and the value provided by the *Client* is inserted. If no existing entry exists then the new entry is inserted.

If an existing entry was replaced successfully then the *StatusCode* shall be *Good_EntryReplaced.* If a new entry was created the *StatusCode* shall be *Good_EntryInserted.* If the *Server* cannot determine whether it replaced or inserted an entry then the *StatusCode* shall be *Good.*

6.8.3.6 Remove functionality

Setting performInsertReplace = REMOVE_4 removes *Structured History Data* such as *Annotations* from the history database at the specified parameters for one or more *Properties* of *HistoricalDataNodes.*

If a *Structure History Data* entry exists at the specified parameters it is deleted. If *Structured History Data* does not already exist at the specified parameters, then the *StatusCode* shall indicate *Bad_NoEntryExists*.

6.8.4 UpdateEventDetails structure

6.8.4.1 UpdateEventDetails structure detail

Table 31 defines the UpdateEventDetails structure.

Table 31 – UpdateEventDetails

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>UpdateEventDetails</td><td>Structure</td><td>The details for insert, replace, and insert/replace history Event updates.</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>Node id of the Object to be updated.</td></tr>
<tr><td>performInsertReplace</td><td>PerformUpdateType</td><td>Value determines which action of insert, replace, or update is performed.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>INSERT_1</td><td>Perform Insert Event (see 6.8.4.2).</td></tr><tr><td>REPLACE_2</td><td>Perform Replace Event (see 6.8.4.3).</td></tr><tr><td>UPDATE_3</td><td>Perform Update Event (see 6.8.4.4).</td></tr></table></td></tr>
<tr><td>filter</td><td>EventFilter</td><td>If the history of Notification conforms to the EventFilter, the history of the Notification is updated.</td></tr>
<tr><td>eventData[]</td><td>HistoryEventFieldList</td><td>List of Event Notifications to be inserted or updated (see 6.5.4 for HistoryEventFieldList definition).</td></tr>
</table>

6.8.4.2 Insert event functionality

This function is intended to insert new entries, e.g., backfilling of historical *Events*.

Setting performInsertReplace = INSERT_1 inserts entries into the *Event* history database for one or more *HistoricalEventNodes*. The *whereClause* parameter of the *EventFilter* shall be empty. The *SelectClause* shall as a minimum provide the following Event fields: *EventType* and Time. It is also recommended that the *SourceNode* and the *SourceName* fields are provided. If one of the required fields is not provided then the *statusCode* shall indicate *Bad_ArgumentsMissing*. If the historian does not support archiving the specified *EventType* then the *statusCode* shall indicate *Bad_TypeDefinitionInvalid*. If the *SourceNode* is not a valid source for *Events* then the related *operationResults entry* shall indicate *Bad_SourceNodeIdInvalid.* If the *Time* does not fall within range that can be stored then the related *operationResults entry* shall indicate *Bad_OutOfRange.* If the *selectClause* does not include fields which are mandatory for the *EventType* then the *statusCode* shall indicate *Bad_ArgumentsMissing.* If the *selectClause* specifies fields which are not valid for the *EventType* or cannot be saved by the historian then the related *operationResults entry* shall indicate *Good_DataIgnored*. Additional information about the ignored fields shall be provided through *DiagnosticInformation* related to the *operationResults*. The symbolicId contains the index of each ignored field separated with a space and the localizedText contains the symbolic names of the ignored fields.

The *EventId* is a *Server* generated opaque value and a *Client* cannot assume that it knows how to create valid *EventIds*. A *Server* shall be able to generate an appropriate default value for the *EventId* field. If a *Client* does specify the *EventId* in the *selectClause* and it matches an existing *Event* then the *statusCode* shall indicate *Bad_EntryExists*. A *Client* shall use a *HistoryRead* to discover any automatically generated *EventIds*.

If any errors occur while processing individual fields then the related o*perationResults entry* shall indicate *Bad_InvalidArgument* and the invalid fields shall be indicated in the *DiagnosticInformation* related to the *operationResults entry*.

The *IndexRange* parameter of the *SimpleAttributeOperand* is not valid for insert operations and the *StatusCode* shall specify Bad_*IndexRangeInvalid* if one is specified.

A *Client* may instruct the *Server* to choose a suitable default value for a field by specifying a value of null. If the *Server* is not able to select a suitable default then the corresponding entry in the operationResults array for the affected *Event* shall be *Bad_InvalidArgument*.

6.8.4.3 Replace event functionality

This function is intended to replace fields in existing *Event* entries, e.g., correct *Event* data that contained incorrect data due to a bad sensor.

Setting performInsertReplace = REPLACE_2 replaces entries in the *Event* history database for the specified *EventIds* for one or more *HistoricalEventNodes*. The *SelectClause* parameter of the *EventFilter* shall specify the *EventId Property* and the *eventData* shall contain the *EventId* which will be used to find the *Event* to be replaced. If no entry exists matching the specified *EventId* then no replace operation will be performed; instead the *operationResults entry* for the *eventData* entry shall indicate *Bad_NoEntryExists*. The *whereClause* parameter of the *EventFilter* shall be empty.

If the *selectClause* specifies fields which are not valid for the *EventType* or cannot be saved or changed by the historian then the *operationResults entry* for the affected *Event* shall indicate *Good_DataIgnored*. Additional information about the ignored fields shall be provided through *DiagnosticInformation* related to the *operationResults*. The symbolicId contains the index of each ignored field separated with a space and the localizedText contains the symbolic names of the ignored fields.

If fatal errors occur while processing individual fields then the *operationResults entry* for the affected Event shall indicate *Bad_InvalidArgument* and the invalid fields shall be indicated in the *DiagnosticInformation* related to the *operationResults entry*.

6.8.4.4 Update event functionality

This function is intended to unconditionally insert/replace *Events*, e.g., synchronizing a backup *Event* database.

Setting performInsertReplace = UPDATE_3 inserts or replaces entries in the *Event* history database for the specified filter for one or more *HistoricalEventNodes*.

The *Server* will, based on its own criteria, attempt to determine if the *Event* already exists; if it does exist then the *Event* will be deleted and the new *Event* will be inserted (retaining the *EventId*). If the *EventID* was provided then the *EventID* will be used to determine if the *Event* already exists. If the *Event* does not exist then a new *Event* will be inserted, including the generation of a new *EventId*.

All of the restrictions, behaviours, and errors specified for the Insert functionality (see 6.8.4.2) also apply to this function.

If an existing *Event* entry was replaced successfully then the related *operationResults entry* shall be *Good_EntryReplaced.* If a new *Event* entry was created then the related *operationResults entry* shall be *Good_EntryInserted.* If the *Server* cannot determine whether it replaced or inserted an entry then the related *operationResults entry*shall be *Good.*

6.8.5 DeleteRawModifiedDetails structure

6.8.5.1 DeleteRawModifiedDetails structure detail

Table 32 defines the DeleteRawModifiedDetails structure.

Table 32 – DeleteRawModifiedDetails

Name	Type	Description
DeleteRawModifiedDetails	Structure	The details for delete raw and delete modified history updates.
nodeId	NodeId	Node id of the *Object* for which history values are to be deleted.
isDeleteModified	Boolean	TRUE for MODIFIED, FALSE for RAW. Default value is FALSE.
startTime	UtcTime	Beginning of period to be deleted.
endTime	UtcTime	End of period to be deleted.

These functions are intended to be used to delete data that has been accidentally entered into the history database, e.g., deletion of data from a source with incorrect timestamps. Both *startTime* and *endTime* shall be defined. The *startTime* shall be less than the *endTime*, and values up to but not including the *endTime* are deleted. It is permissible for *startTime* = *endTime*, in which case the value at the *startTime* is deleted.

6.8.5.2 Delete raw functionality

Setting isDeleteModified = FALSE deletes all *Raw* entries from the history database for the specified time domain for one or more *HistoricalDataNodes*.

If no data is found in the time range for a particular *HistoricalDataNode*, then the *StatusCode* for that item is *Bad_NoData.*

6.8.5.3 Delete modified functionality

Setting isDeleteModified = TRUE deletes all *Modified* entries from the history database for the specified time domain for one or more *HistoricalDataNodes*.

If no data is found in the time range for a particular HistoricalDataNode, then the *StatusCode* for that item is *Bad_NoData.*

6.8.6 DeleteAtTimeDetails structure

6.8.6.1 DeleteAtTimeDetails structure detail

Table 33 defines the structure of the DeleteAtTimeDetails structure.

Table 33 – DeleteAtTimeDetails

Name	Type	Description
DeleteAtTimeDetails	Structure	The details for delete raw history updates
nodeId	NodeId	Node id of the *Object* for which history values are to be deleted.
reqTimes []	UtcTime	The entries define the specific timestamps for which values are to be deleted.

6.8.6.2 Delete at time functionality

The DeleteAtTime structure deletes all entries in the history database for the specified timestamps for one or more *HistoricalDataNodes*.

This parameter is intended to be used to delete specific data from the history database, e.g., lab data that is incorrect and cannot be correctly reproduced.

6.8.7 DeleteEventDetails structure

6.8.7.1 DeleteEventDetails structure detail

Table 34 defines the structure of the DeleteEventDetails structure.

Table 34 – DeleteEventDetails

Name	Type	Description
DeleteEventDetails	Structure	The details for delete raw and delete modified history updates.
nodeId	NodeId	Node id of the *Object* for which history values are to be deleted.
eventId[]	ByteString	An array of *EventIds* to identify which *Events* are to be deleted.

6.8.7.2 Delete event functionality

The DeleteEventDetails structure deletes all *Event* entries from the history database matching the *EventId* for one or more *HistoricalEventNodes*.

If no Events are found that match the specified filter for a *HistoricalEventNode*, then the *StatusCode* for that Node is *Bad_NoData.*

Annex A
(informative)

Client conventions

A.1 How clients may request timestamps

The OPC HDA COM based specifications allowed a *Client* to programmatically request historical time periods as absolute time (Jan 01, 2006 12:15:45) or a string representation of relative time (NOW -5M). The OPC UA specification does not allow for using a string representation to pass date/time information using the standard *Services*.

OPC UA *Client* applications that wish to visually represent date/time in a relative string format shall convert this string format to UTC DateTime values before sending requests to the UA *Server*. It is recommended that all OPC UA *Clients* use the syntax defined in Clause A.1 to represent relative times in their user interfaces.

The format for the relative time is:

```
keyword+/-offset+/-offset…
```

where keyword and offset are as specified in Table A.1 below. Whitespace is ignored. The time string shall begin with a keyword. Each offset shall be preceded by a signed integer that specifies the number and direction of the offset. If the integer preceding the offset is unsigned then the value of the preceding sign is assumed (beginning default sign is positive). The keyword refers to the beginning of the specified time period. DAY means the timestamp at the beginning of the current day (00:00 hours, midnight). MONTH means the timestamp at the beginning of the current month, etc.

For example, “DAY -1D+7H30M” could represent the start time for data requested for a daily report beginning at 7:30 in the morning of the previous day (DAY = the first timestamp for today, -1D would make it the first timestamp for yesterday, +7H would take it to 7 a.m. yesterday, +30M would make it 7:30 a.m. yesterday (the + on the last term is carried over from the last term)).

Similarly, “MONTH-1D+5H” would be 5 a.m. on the last day of the previous month, “NOW-1H15M” would be an hour and fifteen minutes ago, and “YEAR+3MO” would be the first timestamp of April 1 this year.

Resolving relative timestamps is based upon what Microsoft has done with Excel, thus for various questionable time strings we have these results:

10-Jan-2001 + 1 MO = 10-Feb-2001

29-Jan-1999 + 1 MO = 28-Feb-1999

31-Mar-2002 + 2 MO = 30-May-2002

29-Feb-2000 + 1 Y = 28-Feb-2001

In handling a gap in the calendar (due to different numbers of days in the month, or in the year), when one is adding or subtracting months or years:

Month: If the answer falls in the gap then it is backed up to the same time of day on the last day of the month.

Year: If the answer falls in the gap (February 29) then it is backed up to the same time of day on February 28.

Note that the above does not hold true for cases of adding or subtracting weeks or days, but only for adding or subtracting months or years which may have different numbers of days in them.

Note that all keywords and offsets are specified in uppercase (see Tables A.1 and A.2).

Table A.1 – Time keyword definitions

Keyword	Description
NOW	The current UTC time as calculated on the *Server*.
SECOND	The start of the current second.
MINUTE	The start of the current minute.
HOUR	The start of the current hour.
DAY	The start of the current day.
WEEK	The start of the current week.
MONTH	The start of the current month.
YEAR	The start of the current year.

Table A.2 –Time offset definitions

Offset	Description
S	Offset from time in seconds.
M	Offset from time in minutes.
H	Offset from time in hours.
D	Offset from time in days.
W	Offset from time in weeks.
MO	Offset from time in months.
Y	Offset from time in years.

A.2 Determining the first historical data point

In some cases *Servers* are required to return the first available data point for a historical *Node;* Clause A.2 recommends the way that a *Client* should request this information so that Servers can optimize this call, if desired. Although there are multiple calls that could return the first data value, the recommended practice will be to use the *StartOfArchive Property*. If this *Property* isn't available then use the following ReadRawModifiedDetails parameters:

returnBounds=false

numValuesPerNode=1

startTime=DateTime.MinValue+1 second

endTime= DateTime.MinValue

Bibliography

IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model*

IEC 62541-6, *OPC Unified Architecture – Part 6: Mappings*

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

IEC 62541-9, *OPC Unified Architecture – Part 9: Alarms and conditions*

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

ARCHITECTURE UNIFIÉE OPC –

Partie 11: Accès à l'Historique

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62541-11 a été établie par le sous-comité 65E: Les dispositifs et leur intégration dans les systèmes de l'entreprise, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

CDV	Rapport de vote
65E/380/CDV	65E/410/RVC

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Une liste de toutes les parties de la série IEC 62541, publiées sous le titre général *Architecture unifiée OPC*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site Web de l'IEC sous “http://webstore.iec.ch” dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

ARCHITECTURE UNIFIÉE OPC –

Partie 11: Accès à l'Historique

1 Domaine d'application

La présente partie de l'IEC 62541 fait partie d'une série de normes d'Architecture Unifiée OPC globale et définit le *Modèle d'Information* associé à l'Accès à l'Historique. Elle inclut particulièrement des descriptions supplémentaires et complémentaires des *NodeClass* (*Classe de Nœuds*) et des *Attributs* nécessaires pour l'Accès à l'Historique, des *Propriétés* normalisées supplémentaires et d'autres informations et comportements.

Le Modèle d'*Espace d'Adressage* complet comprenant toutes les *NodeClasses* et tous les Attributs est spécifié dans l'IEC 62541-3. Le *Modèle d'Information* prédéfini est présenté dans l'IEC 62541-5. Les *Services* permettant de détecter et d'accéder aux données et événements historiques, ainsi qu'une description des types *ExtensibleParameter* sont spécifiés dans l'IEC 62541-4.

La présente norme inclut une fonctionnalité permettant de calculer et de renvoyer des *Agrégats* (minimum, maximum, average, etc.). Le *Modèle d'Information* et la fonction concrète des *Agrégats* sont définis dans l'IEC 62541-13.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts* (disponible en anglais seulement)

IEC 62541-3*, Architecture unifiée OPC – Partie 3: Modèle de l'Espace d'Adressage*

IEC 62541-4, *Architecture unifiée OPC – Partie 4: Services*

IEC 62541-5, *Architecture unifiée OPC – Partie 5: Modèle d'Informations*

IEC 62541-8, *Architecture unifiée OPC – Partie 8: Accès aux données*

IEC 62541-13, *Architecture unifiée OPC – Partie 13: Agrégats*

3 Termes, définitions et abréviations

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'IEC TR 62541-1, l'IEC 62541-3, l'IEC 62541-4 et l'IEC 62541-13 ainsi que les suivants s'appliquent.

3.1.1
Annotation
métadonnées associées à un élément au niveau d'une instance donnée dans le temps

Note 1 à l'article: Une *Annotation* est une métadonnée associée à un élément au niveau d'une instance donnée dans le temps. Aucune valeur n'est à stocker à cet instant.

3.1.2
BoundingValues
valeurs associées à l'heure de début et à l'heure de fin

Note 1 à l'article: Les *BoundingValues* sont les valeurs qui sont associées à l'heure de début et à l'heure de fin d'un *ProcessingInterval* spécifié lors de la lecture de l'historique. Les *BoundingValues* peuvent être exigées par les *Clients* pour déterminer les valeurs de début et de fin lors de la demande de *données brutes* sur un intervalle de temps. Si une valeur de *données brutes* est présente au point de départ ou d'arrivée, elle est considérée comme une valeur limite, même si elle fait parte de la demande de données. S'il n'existe aucune *donnée brute* au point de départ ou d'arrivée, le *Serveur* détermine la valeur limite, qui peut exiger des données provenant d'un point de données se trouvant à l'extérieur de la plage demandée. Voir 4.4 pour plus de détails sur l'utilisation des *BoundingValues*.

3.1.3
HistoricalNode
Objet, *Variable*, *Propriété* ou *Vue* de l'*Espace d'Adressage* dans lequel le *Client* peut accéder aux données historiques ou aux *Événements*

Note 1 à l'article: Un *HistoricalNode* est un terme utilisé dans le présent document pour représenter un *Objet*, *une Variable, une Propriété* ou une *Vue* de l'*Espace d'Adressage* dont un *Client* peut lire et/ou mettre à jour les données historiques ou les *Événements*. Les termes "historique de *HistoricalNode*" ou "historique d'un *HistoricalNode*" se réfèrent aux données de série chronologique ou aux *Événements* enregistrés pour cet *HistoricalNode*. Le terme *HistoricalNode* fait référence aux *HistoricalDataNode* et aux *HistoricalEventNode*.

3.1.4
HistoricalDataNode
Variable ou *Propriété* de l'*Espace d'Adressage* dans lequel un *Client* peut accéder aux données historiques

Note 1 à l'article: Un *HistoricalDataNode* représente une *Variable* ou une *Propriété* de l'*Espace d'Adressage* dont un *Client* peut lire et/ou mettre à jour les données historiques. "Historique de *HistoricalDataNode*" ou "historique d'un *HistoricalDataNode*" se réfèrent aux données de série chronologique enregistrées pour cet *HistoricalNode*. Exemples de ce type de données:

- données de dispositif (les capteurs de température, par exemple),
- données calculées,
- informations de statut (ouvert/fermé, en déplacement),
- données du système à variation dynamique (les cours de Bourse, par exemple),
- données de diagnostic.

Le terme *HistoricalDataNodes* est utilisé lors du référencement des aspects de la norme qui s'appliquent à l'accès aux données historiques uniquement.

3.1.5
HistoricalEventNode
Objet ou *Vue* de l'*Espace d'Adressage* dans lequel un *Client* peut accéder aux historiques d'*Événements*

Note 1 à l'article: "historique de *HistoricalEventNode*" ou "historique d'un *HistoricalEventNode*" se réfère aux *Événements* de série chronologique enregistrés dans certains systèmes historiques. Exemples de ce type de données:

- *Notifications,*
- *Alarmes* de système,
- *Événements* d'action de l'opérateur,
- déclencheurs de systèmes (de nouveaux ordres à traiter, par exemple).

Le terme *HistoricalEventNode* est utilisé lorsqu'il est fait référence à des aspects de la norme qui s'appliquent à l'accès aux *Événements* historiques uniquement.

3.1.6
valeurs modifiées
valeur d'un *HistoricalDataNode* qui a été modifiée (ou insérée/supprimée manuellement) après avoir été stockée dans l'historique

Note 1 à l'article: Pour certains *Serveurs*, une valeur d'entrée de données de laboratoire n'est pas une *valeur modifiée*, mais si un utilisateur corrige une valeur de laboratoire, la valeur originale est considérée comme une *valeur modifiée* et est renvoyée pendant une requête de *valeurs modifiées*. De même, l'insertion manuelle d'une valeur qui a été ignorée par un système de collecte normalisé peut être considérée comme une *valeur modifiée*. Sauf indication contraire, tous les *Services* historiques fonctionnent sur la valeur en cours ou la plus récente pour l'*HistoricalDataNode* à l'horodatage spécifié. Les demandes de *valeurs modifiées* sont utilisées pour accéder aux valeurs qui ont été remplacées, supprimées ou insérées. Il revient au système de déterminer les valeurs considérées comme étant des *valeurs modifiées*. A chaque fois qu'un *Serveur* modifie les données disponibles pour une entrée dans l'ensemble d'historiques, il doit définir le bit *ExtraData* dans le *StatusCode*.

3.1.7
données brutes
données stockées dans l'historique pour un *HistoricalDataNode*

Note 1 à l'article: Les données peuvent être collectées pour la *DataValue* ou peuvent être un sous-ensemble des données, en fonction de l'historique et des règles de stockage utilisées lors de la sauvegarde des valeurs de l'élément.

3.1.8
StartTime/EndTime
limites d'une requête d'historique qui définissent le domaine temporel

Note 1 à l'article: Pour toutes les requêtes, une valeur située à l'heure de fin du domaine temporel n'est pas incluse dans le domaine. Par conséquent, les requêtes formulées pour les domaines temporels successifs contigus incluent exactement une fois chaque valeur de l'ensemble d'historiques.

3.1.9
TimeDomain
intervalle de temps couvert par une requête particulière ou une réponse

Note 1 à l'article: En général, si l'heure de début est antérieure ou égale à l'heure de fin, le domaine temporel est considéré commencer à l'heure de début et se terminer juste avant l'heure de fin. Si l'heure de fin est antérieure à l'heure de début, le domaine temporel commence toujours à l'heure de début et se termine juste avant l'heure de fin, l'heure "revenant en arrière" pour la requête particulière et la réponse. Dans les deux cas, une valeur qui tombe exactement à l'heure de fin du *TimeDomain* n'est pas incluse dans le *TimeDomain*. Voir les exemples en 4.4. Les *BoundingValues* ont un impact sur le domaine temporel (voir 4.4).

Tous les horodatages qui peuvent légalement être représentés dans un *UtcTime DataType* sont valides, et le *Serveur* peut ne pas renvoyer un code de résultat d'argument non valide, l'horodatage étant hors de la plage pour laquelle le *Serveur* détient des données. Voir l'IEC 62541-3 pour une description de la plage et la granularité de ce *DataType*. Les *Serveurs* sont censés traiter par commande les horodatages hors limites, et renvoyer les bons *StatusCodes* au *Client.*

3.1.10
Données Historiques Structurées
données structurées stockées dans un ensemble d'historiques, dont certaines parties de la structure permettent d'identifier de manière unique les données dans l'ensemble de données

Note 1 à l'article: La plupart des données historiques ne supposent qu'une valeur courante par horodatage. Par conséquent, l'horodatage des données est considéré comme étant l'identificateur unique de cette valeur. Certaines données ou métadonnées (les *Annotations*, par exemple) peuvent admettre plusieurs valeurs au niveau d'un seul horodatage. Dans ce cas, le Serveur utilise un ou plusieurs paramètres de l'entrée de *Données Historiques Structurées* pour identifier de manière unique chaque élément de l'ensemble d'historiques. Les *Annotations* sont des exemples de *Données Historiques Structurées*.

3.2 Abréviations

DA Data Access (Accès aux Données)
HA Historical Access (Accès à l'Historique)
HDA Historical Data Access (Accès aux Données Historiques)

UA Unified Architecture (Architecture Unifiée)

4 Concepts

4.1 Généralités

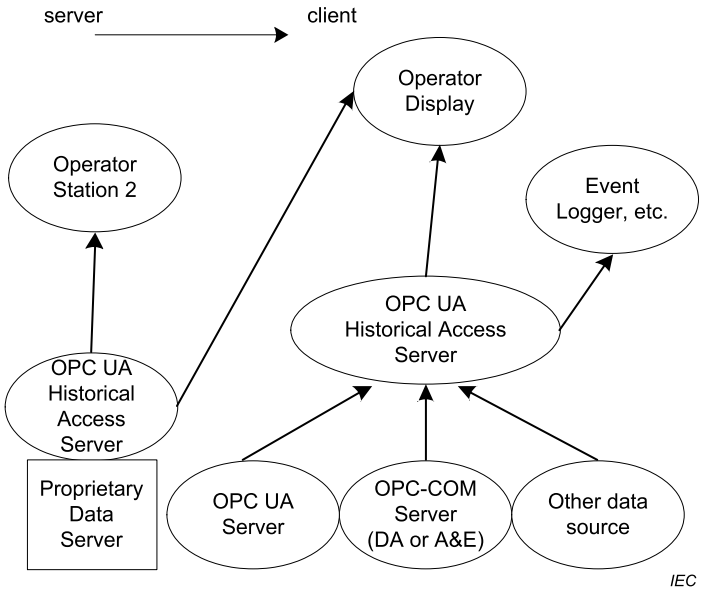
La présente norme définit le traitement des données de série chronologique et les données d'*Événement* historiques dans l'Architecture Unifiée OPC. La spécification de la représentation des données et *Événements* historiques est incluse dans l'*Espace d'Adressage.*

4.2 Architecture de données

Un *Serveur* prenant en charge l'Accès à l'Historique assure aux *Clients* un accès en toute transparence aux différentes données historiques et/ou aux sources d'*Événement* historique (par exemple, historiques de processus, historiques d'événement, etc.).

Les données ou *Événements* historiques peuvent se trouver dans un ensemble de données propriétaires, une base de données ou un tampon à court terme dans la mémoire. Un *Serveur* prenant en charge l'Accès à l'Historique fournit les données et *Événements* historiques pour tout ou un sous-ensemble des *Variables*, *Objets, Propriétés* ou *Vues* disponibles dans l'*Espace d'Adressage* du *Serveur*.

La Figure 1 illustre la manière dont l'*Espace d'Adressage* d'un *Serveur* UA peut être composé d'une large plage de données historiques et/ou sources d'*Événement* historique.



Légende

Anglais	Français
Server	Serveur
Operator station	Station d'opérateur
Operator display	Affichage d'opérateur
Event logger, etc.	Journal d'événements, etc.
OPC UA Historical Access Server	Serveur d'Accès à l'Historique OPC UA
Proprietary Data Server	Serveur de Données propriétaires
OPC UA Server	Serveur OPC UA
OPC-COM Server (DA or A&E)	Serveur OPC-COM (DA ou A&E)
Other data source	Autre source de données

Figure 1 – Serveur OPC UA prenant en charge l'Accès à l'Historique possible

Le *Serveur* peut être mis en œuvre en tant que Serveur OPC UA autonome qui collecte les données depuis un autre *Serveur* OPC UA ou d'une autre source de données. Le *Client* qui fait référence au Serveur OPC UA prenant en charge l'Accès à l'Historique pour les données historiques peut simplement établir la tendance des modules qui souhaitent obtenir des valeurs sur une base de temps donnée ou établir des rapports complexes nécessitant des données en plusieurs formats.

4.3 Horodatages

La nature de l'Accès à l'Historique OPC UA implique de n'utiliser qu'une seule référence d'horodatage pour relier les points de données multiples, le *Client* pouvant demander quel horodatage sera utilisé en référence. Voir l'IEC 62541-4 pour les détails sur l'énumération *TimestampsToReturn*. Un *Serveur* OPC UA prenant en charge l'Accès à l'Historique traite les différents paramétrages d'horodatage comme indiqué ci-dessous. Un *HistoryRead* avec des paramétrages non valides sera rejeté avec un Bad_TimestampsToReturnInvalid. Voir l'IEC 62541-4.

Pour les *HistoricalDataNodes,* le *SourceTimestamp* est utilisé pour déterminer les valeurs de données historiques à renvoyer.

La requête est formulée en termes de *SourceTimestamp*, mais la réponse peut être un *SourceTimestamp*, un *ServerTimestamp* ou les deux horodatages. Si la réponse contient l'horodatage du *Serveur*, l'horodatage peut tomber hors de la plage de temps demandé.

SOURCE_0 Renvoie le *SourceTimestamp*.

SERVER_1 Renvoie le *ServerTimestamp*.

BOTH_2 Renvoie le *SourceTimestamp* et le *ServerTimestamp*.

NEITHER_3 Ce paramétrage n'est pas valide pour un *HistoryRead* qui accède à des *HistoricalDataNodes.*

Dans ce contexte, une référence à des horodatages de cette norme représente soit un *ServerTimestamp* soit un *SourceTimestamp*, selon le type demandé dans le *Service HistoryRead*. Certains *Serveurs* peuvent ne pas prendre en charge l'historisation de *SourceTimestamp* et *ServerTimestamp*, mais tous les *Serveurs* sont censés prendre en charge l'historisation de *SourceTimestamp* (voir l'IEC 62541-7 pour plus de détails sur les *Profils* du Serveur).

Si une requête est formulée pour *ServerTimestam*p et *SourceTimestamp* et que le *Serveur* ne collecte que le *SourceTimestamp*, le *Serveur* doit renvoyer Bad_TimestampsToReturnInvalid.

Pour *HistoricalEventNodes*, ce paramètre ne s'applique pas. Ce paramètre est ignoré étant donné que les entrées sont dictées par le Filtre d'*Événements*. Voir l'IEC 62541-4 pour plus de détails.

4.4 Valeurs limites et domaine temporel

Lors de l'accès aux *HistoricalDataNodes* par l'intermédiaire du *Service HistoryRead*, les requêtes peuvent définir un fanion (returnBounds) indiquant que des *BoundingValues* sont demandées. Pour une description exhaustive du *Paramètre extensible HistoryReadDetails* qui inclut *StartTime*, *EndTime* et NumValuesPerNode, voir 6.4. Le concept de Valeurs limites et la manière dont elles affectent le domaine temporel demandé dans le cadre d'une requête *HistoryRead* est expliqué plus en détail en 4.4. 4.4 donne également des exemples de *TimeDomains* pour illustrer plus en détail le comportement prévu.

Lors de la formulation d'une requête de données historiques à l'aide du *Service HistoryRead*, les paramètres requis incluent au moins 2 de ces trois paramètres: *startTime*, *endTime* et *numValuesPerNode*. Les éléments renvoyés lorsque des Valeurs limites sont demandées varient selon le paramètre fourni parmi ceux ci-dessus. Pour un historique dont les valeurs ont été stockées à 5:00, 5:02, 5:03, 5:05 et 5:06, les données renvoyées lors de l'utilisation de la fonctionnalité Read *Raw* sont données au Tableau 1. Dans le tableau, FIRST indique un uplet avec une valeur nulle, un horodatage de *StartTime* spécifié et un *StatusCode* de *Bad_BoundNotFound*. LAST indique un uplet avec une valeur nulle, un horodatage de *EndTime* spécifié et un *StatusCode* de *Bad_BoundNotFound.*

Dans certains cas, les tentatives de recherche de limites, particulièrement les éléments FIRST et LAST, peuvent obliger le *Serveur* à utiliser beaucoup de ressources. Par conséquent, la mesure dans laquelle les recherches de Valeurs limites ont lieu dans l'historique dépend du *Serveur*, les limites de recherche du *Serveur* pouvant être atteintes avant d'avoir trouvé les valeurs limites. Dans d'autres cas également, comme la lecture de données *Annotations* ou *Attribut*, les Valeurs limites peuvent ne pas être appropriées. Dans ce cas, le *Serveur* peut renvoyer un *StatusCode* de *Bad_BoundNotSupported.*

Tableau 1 – Exemples de Valeur limite

Heure de début	Heure de fin	numValuesPerNode	Limites	Données renvoyées
5:00	05:05	0	Oui	5:00, 5:02, 5:03, 5:05
5:00	05:05	0	Non	5:00, 5:02, 5:03
05:01	05:04	0	Oui	5:00, 5:02, 5:03, 5:05
05:01	05:04	0	Non	5:02, 5:03
05:05	5:00	0	Oui	05:05, 05:03, 05:02, 05:00
05:05	5:00	0	Non	05:05, 05:03, 05:02
05:04	05:01	0	Oui	05:05, 05:03, 05:02, 05:00
05:04	05:01	0	Non	05:03, 05:02
04:59	05:05	0	Oui	FIRST, 5:00, 5:02, 5:03, 5:05
04:59	05:05	0	Non	5:00, 5:02, 5:03
05:01	05:07	0	Oui	5:00, 5:02, 5:03, 5:05, 5:06, LAST
05:01	05:07	0	Non	05:02, 05:03, 05:05, 05:06
5:00	05:05	3	Oui	5:00, 5:02, 5:03
5:00	05:05	3	Non	5:00, 5:02, 5:03
05:01	05:04	3	Oui	5:00, 5:02, 5:03
05:01	05:04	3	Non	5:02, 5:03
05:05	5:00	3	Oui	05:05, 05:03, 05:02
05:05	5:00	3	Non	05:05, 05:03, 05:02
05:04	05:01	3	Oui	05:05, 05:03, 05:02
05:04	05:01	3	Non	05:03, 05:02
04:59	05:05	3	Oui	FIRST, 5:00, 5:02
04:59	05:05	3	Non	5:00, 5:02, 5:03
05:01	05:07	3	Oui	5:00, 5:02, 5:03
05:01	05:07	3	Non	05:02, 05:03, 05:05
5:00	UNSPECIFIED	3	Oui	5:00, 5:02, 5:03
5:00	UNSPECIFIED	3	Non	5:00, 5:02, 5:03
5:00	UNSPECIFIED	6	Oui	5:00, 5:02, 5:03, 5:05, 5:06, LAST[a]
5:00	UNSPECIFIED	6	Non	5:00, 5:02, 5:03, 5:05, 5:06
05:07	UNSPECIFIED	6	Oui	5:06, LAST
05:07	UNSPECIFIED	6	Non	NODATA
UNSPECIFIED	05:06	3	Oui	5:06,5:05,5:03
UNSPECIFIED	05:06	3	Non	5:06,5:05,5:03
UNSPECIFIED	05:06	6	Oui	5:06,5:05,5:03,5:02,5:00,FIRST[b]
UNSPECIFIED	05:06	6	Non	5:06, 5:05, 5:03, 5:02, 5:00
UNSPECIFIED	04:48	6	Oui	5:00, FIRST
UNSPECIFIED	04:48	6	Non	NODATA
04:48	04:48	0	Oui	FIRST,5:00
04:48	04:48	0	Non	NODATA
04:48	04:48	1	Oui	FIRST
04:48	04:48	1	Non	NODATA
04:48	04:48	2	Oui	FIRST,5:00
5:00	5:00	0	Oui	5:00,5:02[c]

Heure de début	Heure de fin	numValuesPerNode	Limites	Données renvoyées
5:00	5:00	0	Non	5:00
5:00	5:00	1	Oui	5:00
5:00	5:00	1	Non	5:00
05:01	05:01	0	Oui	05:00, 05:02
05:01	05:01	0	Non	NODATA
05:01	05:01	1	Oui	5:00
05:01	05:01	1	Non	NODATA

a L'horodatage de LAST ne peut pas être l'Heure de fin spécifiée, car aucune Heure de fin n'est précisée. Dans ce cas, l'horodatage de LAST est égal à l'horodatage précédent renvoyé plus une seconde.

b L'horodatage de FIRST ne peut pas être l'Heure de fin spécifiée, car aucune Heure de début n'est précisée. Dans ce cas, l'horodatage de FIRST est égal à l'horodatage précédent renvoyé moins une seconde.

c Si l'Heure de début est égale à l'Heure de fin (il y a des données à cet instant), et la valeur True est attribuée aux Limites, les limites de début sont égales à l'Heure de début, et le point de données suivant est utilisé pour les limites de fin.

4.5 Modifications de l'Espace d'Adressage dans le temps

Les *Clients* utilisent les *Services* navigation du *View Service Set* (Jeu de services pour les vues) pour naviguer à travers l'*Espace d'Adressage* et découvrir les *HistoricalNodes* et leurs caractéristiques. Ces *Services* offrent les dernières informations relatives à l'*Espace d'Adressage*. Il est possible que l'*Espace d'Adressage* d'un *Serveur* change dans le temps (c'est-à-dire que les *TypeDefinitions* peuvent changer. Les *NodeIds* peuvent être modifiés, ajoutés ou supprimés).

Les développeurs et administrateurs de *Serveur* ont à bien comprendre que la modification de l'*Espace d'Adressage* peut avoir un impact sur l'aptitude d'un *Client* à accéder aux informations historiques. Si l'historique d'un *HistoricalNode* est toujours requis, mais que le *HistoricalNode* n'est plus historisé, il convient de conserver l'*Objet* dans l'Espace d'Adressage, avec les paramètres d'*Attribut AccessLevel* et d'*Attribut Historizing* appropriés (voir l'IEC 62541-3 pour plus de détails sur les niveaux d'accès).

5 Modèle d'Information d'historique

5.1 HistoricalNodes

5.1.1 Généralités

Le modèle d'Accès à l'Historique définit les *Propriétés* applicables pour *HistoricalDataNodes* et *HistoricalEventNodes.*

5.1.2 Propriété d'Annotations

La *DataVariable* ou l'*Objet* contenant des données *Annotation* ajoute la *Propriété d'Annotations* (voir Tableau 2 ci-après).

Tableau 2 – Propriété d'Annotations

Nom	Utilisation	Type de données	Description
Propriétés normalisées			
Annotations	O	Annotation	La *Propriété d'Annotations* est utilisée pour indiquer que les données d'*Annotation* existent pour l'ensemble d'historiques présenté par un *HistoricalDataNode*. *DataType* d'*Annotation* est défini en 5.5

Étant donné que les *Propriétés* ne peuvent pas avoir elles-mêmes de *Propriétés*, la *Propriété* d'*Annotation* est uniquement disponible pour des *DataVariables* ou des *Objets*.

Tous les *HistoricalDataNode* de l'*Espace d'Adressage* ne peuvent pas contenir de données d'*Annotation*. La *Propriété* d'*Annotations* indique si le *HistoricalDataNode* prend en charge ou pas les *Annotations*. Les données d'*Annotation* sont accessibles à l'aide des fonctions *HistoryRead* normalisées. Les *Annotations* sont modifiées, insérées ou supprimées à l'aide des fonctions *HistoryUpdate* normalisées.

Comme avec tous les *HistoricalNodes*, les modifications, suppressions ou ajouts d'*Annotations* augmentent les Événements d'audit historiques avec le *NodeId* correspondant.

5.2 HistoricalDataNodes

5.2.1 Généralités

Le modèle Données historiques définit des *ObjectTypes et Objets* supplémentaires. Ces descriptions incluent également les cas d'utilisation requis pour *HistoricalDataNodes.*

5.2.2 HistoricalDataConfigurationType

Le modèle Données d'Accès à l'Historique étend le modèle de type normalisé en définissant le *HistoricalDataConfigurationType.* Cet *Objet* définit les caractéristiques générales d'un *Nœud* qui définit la configuration historique d'un *HistoricalDataNode* défini pour contenir l'historique. Il est formellement défini au Tableau 3.

Toutes les *Instances* de *HistoricalDataConfigurationType* utilisent le *BrowseName* normalisé défini au Tableau 6.

Tableau 3 – Définition de HistoricalDataConfigurationType

Attribut	**Valeur**				
BrowseName	HistoricalDataConfigurationType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasComponent	Objet	AggregateConfiguration	--	AggregateConfiguration Type	Obligatoire
HasComponent	Objet	AggregateFunctions	--	FolderType	Facultatif
HasProperty	Variable	Échelonné	Booléen	PropertyType	Obligatoire
HasProperty	Variable	Définition	Chaîne	PropertyType	Facultatif
HasProperty	Variable	MaxTimeInterval	Durée	PropertyType	Facultatif
HasProperty	Variable	MinTimeInterval	Durée	PropertyType	Facultatif
HasProperty	Variable	ExceptionDeviation	Double	PropertyType	Facultatif
HasProperty	Variable	ExceptionDeviationFormat	Enum	PropertyType	Facultatif
HasProperty	Variable	StartOfArchive	UtcTime	PropertyType	Facultatif
HasProperty	Variable	StartOfOnlineArchive	UtcTime	PropertyType	Facultatif

AggregateConfiguration Object représente le point d'entrée de navigation des informations relatives à la manière dont le *Serveur* traite la fonctionnalité spécifique à l'*Agrégat* (les *données Uncertain*, par exemple). Cet *Objet* est tenu d'être présent, même s'il ne contient pas d'*Objets* de configuration d'*Agrégat*. Les *Agrégats* sont définis dans l'IEC 62541-13.

AggregateFunctions est un point d'entrée permettant de naviguer vers toutes les capacités d'*Agrégat* prises en charge par le *Serveur* pour l'Accès à l'Historique. Il convient que tous les *HistoryAggregates* pris en charge par le *Serveur* soient accessibles à partir de cet *Objet*. Les *Agrégats* sont définis dans l'IEC 62541-13.

La *Variable échelonnée* spécifie si les données historiques ont été collectées de sorte qu'il convient de les afficher en tant que *SlopedInterpolation* (ligne inclinée entre des points) ou que *SteppedInterpolation* (lignes horizontales connectées verticalement entre des points) lorsque les *données brutes* sont examinées. Cette *Propriété* a également un impact sur la manière de calculer certains *Agrégats*. Une valeur True indique le mode d'interpolation échelonnée. Une valeur False indique le mode *SlopedInterpolation*. La valeur par défaut est False.

La *Variable de définition* est une chaîne lisible par l'homme et spécifique au fournisseur qui indique la manière de calculer la valeur de ce *HistoricalDataNode*. La définition n'est pas localisée et contient souvent une équation qui peut être analysée par certains *Clients*.

Exemple: *Définition*::= “(TempA – 25) + TempB”

La *Variable MaxTimeInterval* spécifie l'intervalle maximal entre des points de données dans le référentiel d'historique, quelle que soit la modification de leur valeur (voir IEC 62541-3 pour la définition de *Durée*).

La *Variable MinTimeInterval* spécifie l'intervalle minimal entre des points de données dans le référentiel d'historique, quelle que soit la modification de leur valeur (voir IEC 62541-3 pour la définition de *Durée*).

La *Variable ExceptionDeviation* spécifie la quantité minimale que les données de l'*HistoricalDataNode* doivent modifier afin de pouvoir consigner la modification dans la base de données historique.

La *Variable ExceptionDeviationFormat* spécifie la manière de déterminer ExceptionDeviation. Sa valeur est définie au Tableau 4.

La *Variable StartOfArchive* spécifie la date avant laquelle il n'y a aucune donnée dans l'archive en ligne ou hors ligne.

La *Variable StartOfOnlineArchive* spécifie la date des données les plus anciennes dans l'archive en ligne.

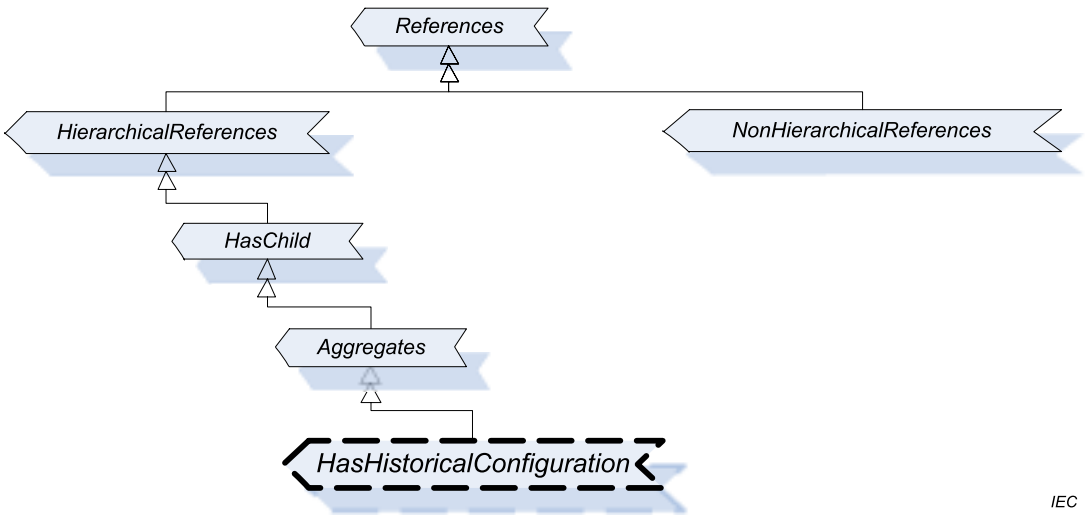
Tableau 4 – Valeurs ExceptionDeviationFormat

Valeur	Description
ABSOLUTE_VALUE_0	ExceptionDeviation est une Valeur absolue.
PERCENT_OF_VALUE_1	ExceptionDeviation est un pourcentage de Valeur.
PERCENT_OF_RANGE_2	ExceptionDeviation est un pourcentage d'InstrumentRange (voir l'IEC 62541-8)
PERCENT_OF_EU_RANGE_3	ExceptionDeviation est un pourcentage d'EURange (voir l'IEC 62541-8)
UNKNOWN_4	Le type ExceptionDeviation est Inconnu ou non spécifié.

5.2.3 ReferenceType HasHistoricalConfiguration

Ce *ReferenceType* est un *ReferenceType* concret qui peut être utilisé directement. Il s'agit d'un sous-type du *ReferenceType* des *Agrégats*, et est utilisé pour faire une référence à partir d'un *Nœud* Historique vers un ou plusieurs *Objets HistoricalDataConfigurationType*.

La sémantique indique que le *Nœud* cible est "utilisé" par le *Nœud* source de la *Référence*. La Figure 2 décrit de manière informelle l'emplacement de ce *ReferenceType* dans la hiérarchie OPC UA. Sa représentation dans l'*Espace d'Adressage* est spécifiée au Tableau 5.



Légende

Anglais	Français
References	Références
Aggregates	Agrégats

Figure 2 – Hiérarchie ReferenceType

Tableau 5 – ReferenceType HasHistoricalConfiguration

Attributs	**Valeur**		
BrowseName	HasHistoricalConfiguration		
InverseName	HistoricalConfigurationOf		
Symmetric	False		
IsAbstract	False		
Références	**NodeClass**	**BrowseName**	**Observations**
Le sous-type des Agrégats *ReferenceType* est défini dans l'IEC 62541-5.			

5.2.4 Objet de configuration des données historiques

Cet *Objet* est utilisé comme point d'entrée de navigation des informations relatives à la configuration *HistoricalDataNode*. Le contenu de cet *Objet* est déjà défini par sa définition de type au Tableau 3. Il est formellement défini au Tableau 6. Si une configuration est définie dans *HistoricalDataNode*, une instance doit comporter un *BrowseName* "Configuration HA". Des configurations supplémentaires peuvent être définies avec différents *BrowseNames*. Tous les *Objets* Configuration Historique doivent être référencés à l'aide du *ReferenceType HasHistoricalConfiguration*. Il est également vivement recommandé de choisir les noms d'affichage de manière à clairement décrire la configuration historique ("1 deuxième collection", "Configuration à long terme", par exemple).

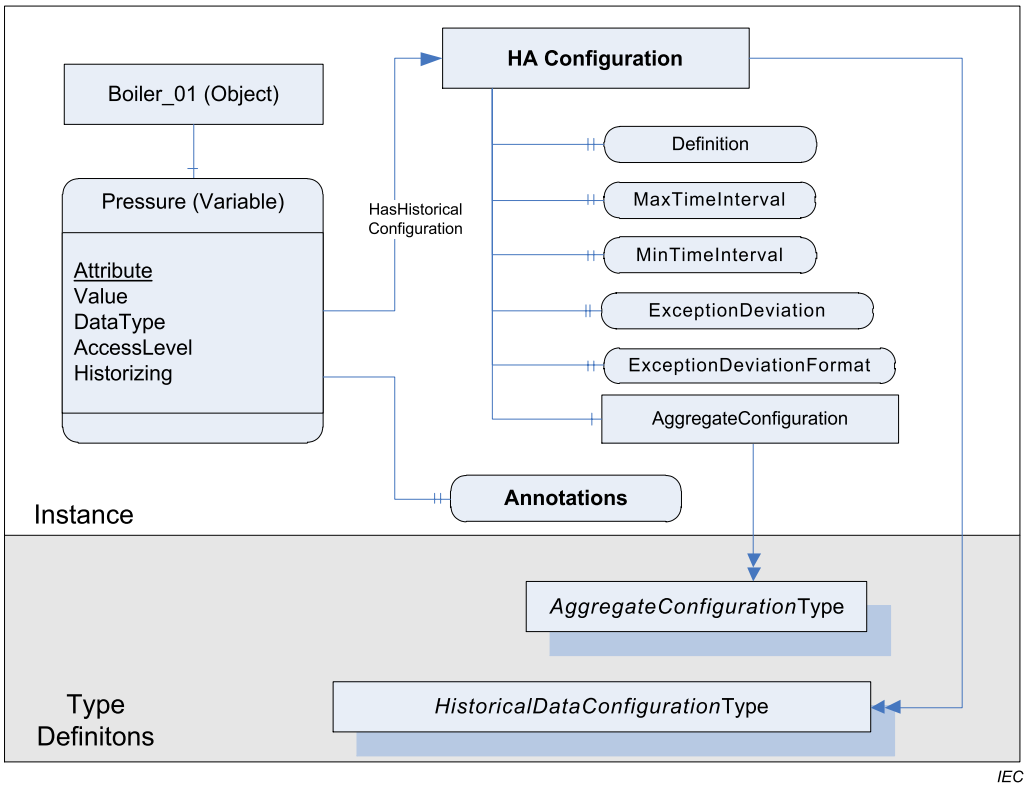
Tableau 6 – Définition de la configuration de l'Accès à l'Historique

Attribut	**Valeur**				
BrowseName	Configuration HA				
Références	**Nœud Classe**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
HasTypeDefinition	Objet Type	HistoricalDataConfigurationType	Défini au Tableau 3		

5.2.5 Modèle d'Espace d'Adressage HistoricalDataNodes

Les *HistoricalDataNodes* font toujours partie d'autres *Nœuds* dans l'Espace d'Adressage. Ils ne sont jamais définis par eux-mêmes. Un "Objet Dossiers" est un exemple simple de conteneur de *HistoricalDataNodes*.

La Figure 3 illustre le Modèle d'*Espace d'Adressage* de base d'une *DataVariable* qui contient l'Historique.



Légende

Anglais	Français
(Object)	(Objet)
HA Configuration	Configuration HA
Attribute	Attribut
Value	Valeur
HasHistorical Configuration	Configuration HasHistorical
Definition	Définition
Value	Valeur
Instance	Instance
Annotations	Annotations
Type Definitions	Définitions de Type

Figure 3 – Variable historique avec configuration et Annotations de données historiques

L'*Attribut Historizing* de chaque *HistoricalDataNode* avec historique (voir l'IEC 62541-3*)* doit être défini et peut faire référence à un *Objet HistoricalAccessConfiguration*. Si HistoricalDataNode est lui-même une *Propriété*, *HistoricalDataNode* hérite des valeurs du Parent de la *Propriété*.

Toutes les *Variables* de l'*Espace d'Adressage* peuvent ne pas contenir de données historiques. Pour savoir si des données historiques sont disponibles, un *Client* recherche les états *HistoryRead*/Write dans l'*Attribut AccessLevel* (voir l'IEC 62541-3 pour plus de détails sur l'utilisation de cet *Attribut).*

La Figure 3 présente uniquement un sous-ensemble d'*Attributs* et de *Propriétés*. Les autres *Attributs* qui sont définis pour les *Variables* dans l'IEC 62541-3 peuvent également être disponibles.

5.2.6 Attributs

Le paragraphe 5.2.6 répertorie les *Attributs* des *Variables* ayant une importance particulière pour les données historiques. Ils sont spécifiés en détail dans l'IEC 62541-3.

- AccessLevel,
- *Historizing.*

5.3 HistoricalEventNodes

5.3.1 Généralités

Le modèle *Événement* historique définit des *Propriétés* supplémentaires. Ces descriptions incluent également les cas d'utilisation requis pour *HistoricalEventNodes.*

L'Accès à l'Historique des *Événements* utilise un *EventFilter*. Il est primordial de bien comprendre les différences entre l'application d'un *EventFilter* à des *Notifications d'événement* en cours et l'extraction d'*Événement* historique.

Dans la surveillance en temps réel, les *Événements* sont reçus par l'intermédiaire de *Notifications* lors de l'abonnement à un *EventNotifier*. L'*EventFilter* assure le filtrage et la sélection de contenu des *Abonnements aux Événements*. Si une *Notification d'événement* est conforme au filtre défini par le paramètre *where* de l'*EventFilter*, la *Notification* est envoyée au *Client*.

Dans l'extraction d'*Événement* historique, l'*EventFilter* représente le filtrage et la sélection de contenu utilisés pour décrire les paramètres des *Événements* disponibles dans l'historique. Ils peuvent ou peuvent ne pas inclure la totalité des paramètres de l'*Événement* en temps réel, c'est-à-dire que tous les champs disponibles lors de la génération de l'*Événement* peuvent ne pas être stockés dans l'historique.

L'*HistoricalEventFilter* peut changer dans le temps. Un *Client* peut donc spécifier un champ d'un *EventType* de l'*EventFilter*. Si un champ n'est pas stocké dans l'ensemble d'historiques, une valeur nulle est attribuée au champ lorsqu'il est référencé dans les paramètres *selectClause* ou *whereClause*.

5.3.2 Propriété HistoricalEventFilter

Un *HistoricalEventNode* contenant un historique d'*Événement* fournit la *Propriété*. Cette *Propriété* est formellement définie au Tableau 7.

Tableau 7 – Propriétés des événements historiques

Nom	Utilisation	Type de données	Description
Propriétés normalisées			
HistoricalEventFilter	O	EventFilter	Filtre utilisé par le *Serveur* pour déterminer les champs *HistoricalEventNode* disponibles dans l'historique. Il peut également inclure une clause where qui indique les types d'*Événements* ou de restrictions sur les *Événements* disponibles par l'intermédiaire de l'*HistoricalEventNode.* La *Propriété HistoricalEventFilter* peut permettre de savoir les champs *Événements* que l'historique stocke. Mais ce champ peut n'avoir aucune incidence sur les champs *Événements* que l'historique est capable de stocker.

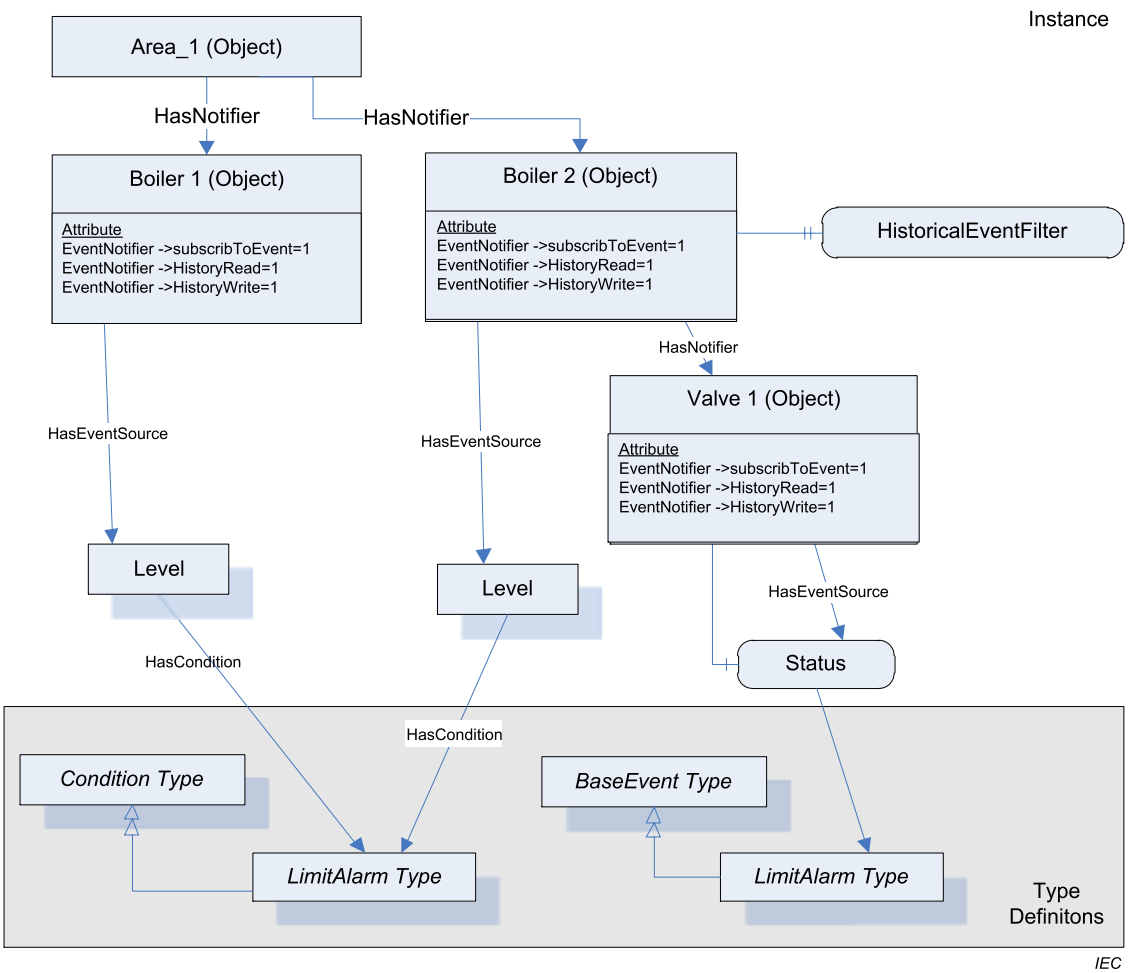
5.3.3 Modèle d'Espace d'Adressage HistoricalEventNodes

Les *HistoricalEventNodes* sont des *Objets* ou des *Vues* de l'*Espace d'Adressage* qui présentent les *Événements* historiques. Ces *Nœuds* sont identifiés grâce à l'*Attribut EventNotifier* et fournissent un sous-ensemble historique des *Événements* générés par le *Serveur*.

Chaque *HistoricalEventNode* est représenté par un *Objet* ou une *Vue* avec un ensemble spécifique d'*Attributs*. La *Propriété HistoricalEventFilter* spécifie les champs disponibles dans l'historique.

Tous les *Objets* ou toutes les *Vues* de l'*Espace d'Adressage* peuvent ne pas être un *HistoricalEventNode*. Pour être considéré comme des *HistoricalEventNodes*, un *Nœud* est tenu de contenir des *Événements* historiques. Pour savoir si les *Événements* historiques sont disponibles, un *Client* recherche les états *HistoryRead*/Write dans l'*Attribut EventNotifier*. Voir l'IEC 62541-3 pour plus de détails sur l'utilisation de cet *Attribut.*

La Figure 4 illustre le Modèle d'*Espace d'Adressage* de base d'un *Événement* qui contient l'Historique.



Légende

Anglais	Français
(Object)	(Objet)
Attribute	Attribut
Instance	Instance

Anglais	Français
Status	Statut
Boiler	Chaudière
Valve	Vanne
Level	Niveau
Type Definitions	Définitions de type

Figure 4 – Représentation d'un Événement avec Historique dans l'Espace d'Adressage

5.3.4 Attributs HistoricalEventNodes

Le paragraphe 5.3.4 répertorie les *Attributs* des *Objets* ou des *Vues* ayant une importance particulière pour les *Événements* historiques. Ils sont spécifiés en détail dans l'IEC 62541-3. Les *Attributs* suivants sont particulièrement importants pour les *HistoricalEventNodes*.

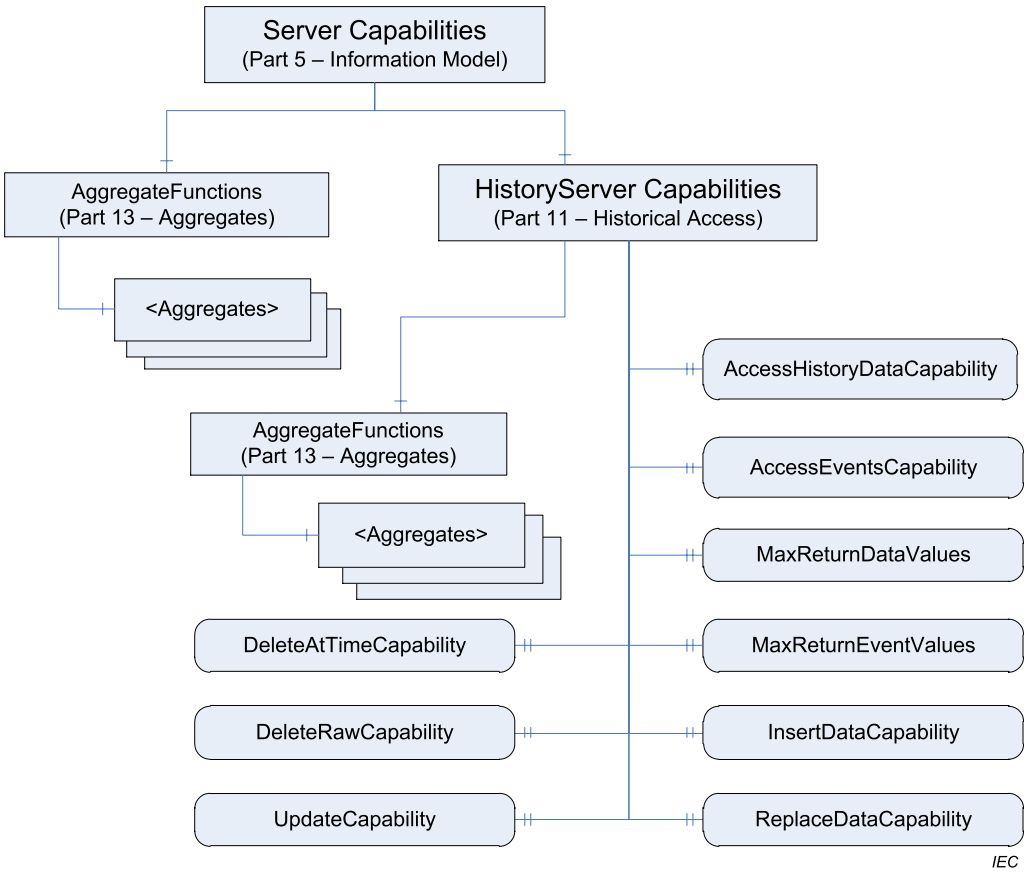
- EventNotifier

L'*Attribut EventNotifier* est utilisé pour indiquer si le *Nœud* peut être utilisé pour lire et/ou mettre à jour les *Événements* historiques.

5.4 Fonctions et capacités de présentation prises en charge

5.4.1 Généralités

Les Serveurs OPC UA peuvent prendre en charge plusieurs fonctionnalités et capacités différentes. Les *Objets* normalisés suivants permettent de présenter ces capacités en commun, plusieurs concepts définis normalisés pouvant être étendus par les fournisseurs. Les *Objets* sont présentés dans l'IEC TR 62541-1.



Légende

Anglais	Français
Server Capabilities (Part 5 – Information Model)	Capacités du *Serveur* (Partie 5 – Modèle d'Information)
Part 13 – Aggregates	Partie 13 – Agrégats
HistoryServer Capabilities (Part 11 – Historical Access)	Capacités HistoryServer (Partie 11 – Accès à l'Historique)
Aggregates	Agrégats

Figure 5 – Serveur et Capacités HistoryServer

5.4.2 HistoryServerCapabilitiesType

Les *Objets ServerCapabilitiesType* d'un Serveur OPC UA prenant en charge l'Accès à l'Historique doivent contenir une *Référence* à un *Objet HistoryServerCapabilitiesType*.

Le contenu de ce *BaseObjectType* est déjà défini par sa définition de type dans l'IEC 62541-5. Les extensions d'*Objet* sont définies de manière formelle au Tableau 8.

Ces propriétés sont destinées à informer un *Client* des capacités générales du *Serveur*. Elles ne garantissent pas la disponibilité de toutes les capacités pour tous les *Nœuds*. Par exemple, tous les *Nœuds* ne prennent pas en charge les *Événements* ou dans le cas d'un *Serveur* de regroupement dans lequel les *Serveurs* sous-jacents peuvent ne pas prendre en charge l'*Insertion* ou un *Agrégat* particulier. Dans ces cas précis, la *Propriété HistoryServerCapabilities* indique que la capacité est prise en charge, et que le *Serveur* renvoie les *StatusCodes* appropriés pour les situations dans lesquelles la capacité ne s'applique pas.

Tableau 8 – Définition de HistoryServerCapabilitiesType

Attribut	**Valeur**				
BrowseName	HistoryServerCapabilitiesType				
IsAbstract	False				
Références	**NodeClass**	**Nom de navigation**	**Type de données**	**Définition de type**	**ModelingRule**
HasProperty	Variable	AccessHistoryDataCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	AccessHistoryEventsCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	MaxReturnDataValues	UInt32	PropertyType	Obligatoire
HasProperty	Variable	*MaxReturnEventValues*	UInt32	PropertyType	Obligatoire
HasProperty	Variable	InsertDataCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	ReplaceDataCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	UpdateDataCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	DeleteRawCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	DeleteAtTimeCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	InsertEventCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	ReplaceEventCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	UpdateEventCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	DeleteEventCapability	Booléen	PropertyType	Obligatoire
HasProperty	Variable	InsertAnnotationsCapability	Booléen	PropertyType	Obligatoire
HasComponent	Objet	AggregateFunctions	--	FolderType	Obligatoire

Tous les Serveurs UA qui prennent en charge l'Accès à l'Historique doivent inclure l'*HistoryServerCapabilities* comme partie intégrante de leurs *ServerCapabilities*.

La *Variable AccessHistoryDataCapability* définit si le *Serveur* prend en charge l'accès aux valeurs de données historiques. Une valeur True indique que le *Serveur* prend en charge l'Accès à l'Historique pour les *HistoricalNodes*, une valeur False indiquant que le *Serveur* ne prend pas en charge l'Accès à l'Historique pour les *HistoricalNodes*. La valeur par défaut est False. La valeur True doit être attribuée à au moins *AccessHistoryDataCapability* ou *AccessHistoryEventsCapability* pour que le *Serveur* soit un Serveur OPC UA prenant en charge l'Accès à l'Historique valide.

La *Variable AccessHistoryEventCapability* définit si le *Serveur* prend en charge l'accès aux *Événements* historiques. Une valeur True indique que le *Serveur* prend en charge l'Accès à l'Historique des *Événements*, une valeur False indiquant que le *Serveur* ne prend pas en charge l'Accès à l'Historique pour les *Événements*. La valeur par défaut est False. La valeur True doit être attribuée à au moins *AccessHistoryDataCapability* ou *AccessHistoryEventsCapability* pour que le *Serveur* soit un Serveur OPC UA prenant en charge l'Accès à l'Historique valide.

La *Variable MaxReturnDataValues* définit le nombre maximal de valeurs que le *Serveur* peut renvoyer pour chaque *HistoricalNode* consulté lors d'une requête. Une valeur 0 indique que le *Serveur* n'impose aucune limite quant au nombre de valeurs qu'il peut renvoyer. Il est valide pour un *Serveur* de limiter le nombre de valeurs renvoyées et de renvoyer un point de continuation même si *MaxReturnValues* = 0. Par exemple, il est possible que le système sous-jacent impose une limite dont le *Serveur* n'a pas connaissance, même si ce dernier n'impose aucune restriction. La valeur par défaut est 0.

De la même manière, *MaxReturnEventValues* spécifie le nombre maximal d'*Événements* qu'un *Serveur* peut renvoyer pour un *HistoricalEventNode*.

La *Variable InsertDataCapability* indique la prise en charge de la capacité d'Insertion. Une valeur True indique que le *Serveur* prend en charge la capacité d'insertion de nouvelles valeurs de données dans l'historique, mais n'écrase pas les données existantes. La valeur par défaut est False.

La *Variable ReplaceDataCapability* indique la prise en charge de la capacité de Remplacement. Une valeur True indique que le *Serveur* prend en charge la capacité de remplacement des valeurs de données existantes dans l'historique, mais n'insère pas de nouvelles données. La valeur par défaut est False.

La *Variable UpdateDataCapability* indique la prise en charge de la capacité de Mise à jour. Une valeur True indique que le *Serveur* prend en charge la capacité d'insertion de nouvelles valeurs de données dans l'historique et remplace les valeurs qui existent. La valeur par défaut est False.

La *Variable DeleteRawCapability* indique la prise en charge de la capacité de suppression de valeurs brutes. Une valeur True indique que le *Serveur* prend en charge la capacité de suppression des valeurs de *données brutes* dans l'historique. La valeur par défaut est False.

La *Variable DeleteAtTimeCapability* indique la prise en charge de la capacité de suppression à temps. Une valeur True indique que le *Serveur* prend en charge la capacité de suppression d'une valeur de données à un moment spécifié. La valeur par défaut est False.

La *Variable InsertEventCapability* indique la prise en charge de la capacité d'Insertion. Une valeur True indique que le *Serveur* prend en charge la capacité d'insertion de nouveaux *Événements* dans l'historique. Une insertion n'est pas un remplacement. La valeur par défaut est False.

La *Variable ReplaceEventCapability* indique la prise en charge de la capacité de Remplacement. Une valeur True indique que le *Serveur* prend en charge la capacité de remplacement d'*Événements* existants dans l'historique. Un remplacement n'est pas une insertion. La valeur par défaut est False.

La *Variable UpdateEventCapability* indique la prise en charge de la capacité de Mise à jour. Une valeur True indique que le *Serveur* prend en charge la capacité d'insertion de nouveaux *Événements* dans l'historique et remplace les valeurs qui existent. La valeur par défaut est False.

La *Variable DeleteEventCapability* indique la prise en charge de la capacité de suppression des *Événements*. Une valeur True indique que le *Serveur* prend en charge la capacité de suppression d'*Événements* dans l'historique. La valeur par défaut est False.

La *Variable InsertAnnotationCapability* indique la prise en charge des *Annotations*. Une valeur True indique que le *Serveur* prend en charge la capacité d'insertion d'*Annotations*. Certains *Serveurs* qui prennent en charge l'insertion d'*Annotations* prennent également en charge l'édition et la suppression des *Annotations*. La valeur par défaut est False.

AggregateFunctions est un point d'entrée permettant de naviguer vers toutes les capacités d'*Agrégat* prises en charge par le *Serveur* pour l'Accès à l'Historique. Il convient que tous les *HistoryAggregates* pris en charge par le *Serveur* soient accessibles à partir de cet *Objet*. Les *Agrégats* sont définis dans l'IEC 62541-13. Si le *Serveur* ne prend pas en charge les *Agrégats*, le *Dossier* reste vide.

5.5 Définitions de DataType historiques

Ce *DataType* décrit les informations d'*Annotation* pour les éléments de données historiques. Ses éléments sont définis au Tableau 9.

Tableau 9 – Structure d'Annotation

Nom	Type	Description
Annotation	Structure	
message	Chaîne	Message ou texte d'*Annotation.*
username	Chaîne	Utilisateur qui a ajouté l'*Annotation*, tel que fourni par le système sous-jacent.
AnnotationTime	UtcTime	Heure à laquelle l'*Annotation* a été ajoutée. Elle sera probablement différente de *SourceTimestamp.*

5.6 Événements d'audit historique

5.6.1 Généralités

Les *AuditEvents* sont générés suite à une action réalisée sur le *Serveur* par un *Client* du *Serveur*. Par exemple, en réponse à un *Client* qui écrit à une *Variable*, le *Serveur* génère un *AuditEvent* décrivant la *Variable* comme étant la source, et la *Session Client* comme étant à l'origine de l'*Événement*. Tous les *Serveurs* ne prennent pas en charge l'audit, mais si c'est le cas d'un *Serveur*, il doit prendre en charge les *Événements* d'audit comme indiqué en 5.6. Les *Profils* (voir l'IEC 62541-7) peuvent être utilisés pour déterminer si un *Serveur* prend en charge l'audit. Les *Serveurs* doivent générer des *Événements* de l'*AuditHistoryUpdateEventType* ou d'un sous-type de ce type pour tous les appels du *Service HistoryUpdate* sur un *HistoricalNode*. Voir l'IEC 62541-3 et l'IEC 62541-5 pour plus de détails sur le modèle *AuditHistoryUpdateEventType*. Si le *Service HistoryUpdate* est appelé pour insérer des *Événements* Historiques, l'*Événement AuditHistoryEventUpdateEventType* doit inclure l'EventId de l'*Événement* inséré et une description indiquant que l'*Événement* a été inséré. Si le *Service HistoryUpdate* est appelé pour supprimer des enregistrements, l'*AuditHistoryDeleteEventType* ou l'un de ses sous-types doit être généré. Voir 6.7 pour plus de détails sur la mise à jour des données ou *Événements* historiques.

En particulier, l'utilisation de la fonctionnalité Delete raw ou Delete modified doit générer un *Événement* AuditHistoryRawModifyDeleteEventType ou l'un de ses sous-types. L'utilisation de la fonctionnalité Delete at time doit générer un *Événement* AuditHistoryAtTimeDeleteEventType ou l'un de ses sous-types. L'utilisation de la fonctionnalité Delete *Event* doit générer un *Événement AuditHistoryEventDeleteEventType* ou l'un de ses sous-types. Toutes les autres mises à jour doivent suivre les lignes directrices fournies dans le modèle *AuditHistoryUpdateEventType*.

5.6.2 AuditHistoryEventUpdateEventType

Il s'agit d'un sous-type d'*AuditHistoryUpdateEventType* utilisé pour classer les *Événements* liés à la mise à jour de l'*Événement* historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 10.

Tableau 10 – Définition d'AuditHistoryEventUpdateEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryEventUpdateEventType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryUpdateEventType* défini dans l'IEC 62541-3, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Obligatoire
HasProperty	Variable	PerformInsertReplace	PerformUpdateType	PropertyType	Obligatoire
HasProperty	Variable	Filtre	EventFilter	PropertyType	Obligatoire
HasProperty	Variable	NewValues	HistoryEventFieldList [ ]	PropertyType	Obligatoire
HasProperty	Variable	OldValues	HistoryEventFieldList [ ]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditHistoryUpdateEventType*. Leur sémantique est définie dans l'IEC 62541-5.

L'*UpdateNode* identifie l'*Attribut* qui était écrit sur le *SourceNode*.

L'énumération *PerformInsertReplace* reflète le paramètre sur l'appel de *Service*.

Le *Filtre* reflète le filtre d'*Événements* passé sur l'appel pour sélectionner les *Événements* qui sont à mettre à jour.

Les *NewValues* identifient la valeur qui était écrite à l'*Événement*.

Les *OldValues* identifient la valeur que les *Événements* contenaient avant la mise à jour. Il est acceptable qu'un *Serveur* qui ne détient pas ces informations signale une valeur nulle. En cas d'insertion, la valeur est censée être nulle.

NewValues et *OldValues* contiennent des *Événements* avec les champs appropriés, contenant chacun les valeurs codées appropriées.

5.6.3 AuditHistoryValueUpdateEventType

Il s'agit d'un sous-type d'*AuditHistoryUpdateEventType* utilisé pour classer les *Événements* liés à la mise à jour de la valeur historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 11.

Tableau 11 – Définition d'AuditHistoryValueUpdateEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryValueUpdateEventType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryUpdateEventType* défini dans l'IEC 62541-3, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Obligatoire
HasProperty	Variable	PerformInsertReplace	PerformUpdateTyp e	PropertyType	Obligatoire
HasProperty	Variable	NewValues	DataValue[]	PropertyType	Obligatoire
HasProperty	Variable	OldValues	DataValue[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditHistoryUpdateEventType*. Leur sémantique est définie dans l'IEC 62541-5.

L'*UpdatedNode* identifie l'*Attribut* qui était écrit sur le *SourceNode*.

L'énumération *PerformInsertReplace* reflète le paramètre sur l'appel de *Service*.

Les *NewValues* identifient la valeur qui était écrite à l'*Événement*.

Les *OldValues* identifient la valeur que l'*Événement* contenait avant l'écriture. Il est acceptable qu'un *Serveur* qui ne détient pas ces informations signale une valeur nulle. En cas d'insertion, la valeur est censée être nulle.

NewValues et *OldValues* contiennent une valeur dans le *DataType* et le codage utilisé pour écrire la valeur.

5.6.4 AuditHistoryDeleteEventType

Il s'agit d'un sous-type d'*AuditHistoryUpdateEventType* utilisé pour classer les *Événements* liés à la suppression de l'historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 12.

Tableau 12 – Définition d'AuditHistoryDeleteEventType

Attribut	**Valeur**				
BrowseName	*AuditHistoryDeleteEventType*				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryUpdateEventType* défini dans l'IEC 62541-3, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	UpdatedNode	NodeId	PropertyType	Obligatoire
HasSubtype	ObjectType	AuditHistoryRawModifyDeleteEventType			
HasSubtype	ObjectType	AuditHistoryAtTimeDeleteEventType			
HasSubtype	ObjectType	AuditHistoryEventDeleteEventType			

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditUpdateEventType*. Leur sémantique est définie dans l'IEC 62541-5.

Le *NodeId* identifie le *NodeId* qui était utilisé pour l'opération de suppression.

5.6.5 AuditHistoryRawModifyDeleteEventType

Il s'agit d'un sous-type d'*AuditHistoryDeleteEventType* utilisé pour classer les *Événements* liés à la suppression de l'historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 13.

Tableau 13 – Définition d'AuditHistoryRawModifyDeleteEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryRawModifyDeleteEventType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryDeleteEventType* défini au Tableau 12, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	IsDeleteModified	Booléen	PropertyType	Obligatoire
HasProperty	Variable	*StartTime*	UtcTime	PropertyType	Obligatoire
HasProperty	Variable	*EndTime*	UtcTime	PropertyType	Obligatoire
HasProperty	Variable	OldValues	DataValue[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditHistoryDeleteEventType*. Leur sémantique est définie en 5.6.4.

L'*isDeleteModified* reflète le paramètre isDeleteModified de l'appel.

La *StartTime* reflète le paramètre d'heure de début de l'appel.

L'*EndTime* reflète le paramètre d'heure de fin de l'appel.

Les *OldValues* identifient la valeur que l'historique contenait avant la suppression. Il convient qu'un *Serveur* signale toutes les valeurs supprimées. Il est acceptable qu'un *Serveur* qui ne détient pas ces informations signale une valeur nulle. Les *OldValues* contiennent une valeur dans le *DataType* et le codage utilisé pour écrire la valeur.

5.6.6 AuditHistoryAtTimeDeleteEventType

Il s'agit d'un sous-type d'*AuditHistoryDeleteEventType* utilisé pour classer les *Événements* liés à la suppression de l'historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 14.

Tableau 14 – Définition d'AuditHistoryAtTimeDeleteEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryAtTimeDeleteEventType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryDeleteEventType* défini au Tableau 12, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	ReqTimes	UtcTime[]	PropertyType	Obligatoire
HasProperty	Variable	OldValues	DataValues[]	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditHistoryDeleteEventType*. Leur sémantique est définie en 5.6.7.

Les *ReqTimes* reflètent le paramètre d'heure de la requête de l'appel.

Les *OldValues* identifient la valeur que l'historique contenait avant la suppression. Il convient qu'un *Serveur* signale toutes les valeurs supprimées. Il est acceptable qu'un *Serveur* qui ne détient pas ces informations signale une valeur nulle. Les *OldValues* contiennent une valeur dans le *DataType* et le codage utilisé pour écrire la valeur.

5.6.7 AuditHistoryEventDeleteEventType

Il s'agit d'un sous-type d'*AuditHistoryDeleteEventType* utilisé pour classer les *Événements* liés à la suppression de l'historique. Ce type suit le comportement de son type parent. Sa représentation dans l'*Espace d'Adressage* est définie de manière formelle au Tableau 15.

Tableau 15 – Définition d'AuditHistoryEventDeleteEventType

Attribut	**Valeur**				
BrowseName	AuditHistoryEventDeleteEventType				
IsAbstract	False				
Références	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Sous-type d'*AuditHistoryDeleteEventType* défini au Tableau 12, c'est-à-dire que les *Références HasProperty* sont attribuées aux mêmes *Nœuds*.					
HasProperty	Variable	EventIds	ByteString[]	PropertyType	Obligatoire
HasProperty	Variable	OldValues	HistoryEventFieldList	PropertyType	Obligatoire

Cet *EventType* hérite de toutes les *Propriétés* de l'*AuditHistoryDeleteEventType*. Leur sémantique est définie en 5.6.4.

Les *EventIds* reflètent le paramètre EventIds de l'appel.

Les *OldValues* identifient la valeur que l'historique contenait avant la suppression. Il convient qu'un *Serveur* signale toutes les valeurs supprimées. Il est acceptable qu'un *Serveur* qui ne détient pas ces informations signale une valeur nulle. Les *OldValues* contiennent un *Événement* avec les champs appropriés, contenant chacun les valeurs codées appropriées.

6 Utilisation des Services spécifique à l'Accès à l'Historique

6.1 Généralités

L'IEC 62541-4 spécifie tous les *Services* nécessaires pour l'Accès à l'Historique OPC UA. En particulier:

- Le Jeu de service de navigation ou le Jeu de service d'interrogation afin de détecter les *HistoricalNodes* et leur configuration.
- Les Services *HistoryRead* et *HistoryUpdate* du Jeu de services d'attributs pour lire et mettre à jour l'historique des *HistoricalNodes*.

6.2 StatusCodes des Nœuds Historiques

6.2.1 Vue d'ensemble

Le 6.2 définit les codes et règles supplémentaires qui s'appliquent au *StatusCode* utilisé pour les *HistoricalNodes*.

La structure générale du *StatusCode* est spécifiée dans l'IEC 62541-4. Elle inclut un ensemble de codes de résultat opérationnel communs qui s'appliquent également aux données et/ou *Événements* historiques.

6.2.2 Codes de résultat de niveau d'opération

Dans l'Accès à l'Historique OPC UA, le *StatusCode* permet d'indiquer les conditions dans lesquelles une *Valeur* ou un *Événement* a été stocké(e) et, de ce fait, peut être utilisé(e) comme un indicateur de validité. Compte tenu de la nature des données et/ou *Événements* historiques, des informations supplémentaires, outre la qualité de base et le code de résultat d'appel, sont à envoyer au *Client*, par exemple, si la valeur est stockée dans le référentiel de

données, si le résultat a été *Interpolé*, si toutes les entrées de données dans un calcul étaient de bonne qualité, etc.

Le Tableau 16 ci-dessous contient les codes avec la sévérité *Bad* indiquant une défaillance. Le Tableau 17 contient les codes *Good* (succès).

Il est important de noter qu'il s'agit de codes spécifiques à l'Accès à l'Historique OPC UA, qui complètent ceux qui s'appliquent à tous les types de données, et qui sont donc définis dans l'IEC 62541-4, l'IEC 62541-8 et l'IEC 62541-13.

Tableau 16 – Codes de résultat de niveau d'opération Bad

ID symbolique	Description
Bad_NoData	Il n'existe aucune donnée pour l'intervalle de temps ou le filtre d'*Événements* demandé
Bad_BoundNotFound	Pas de donnée pour fournir la valeur limite supérieure ou inférieure.
Bad_BoundNotSupported	Les Valeurs limites ne sont pas applicables ou le *Serveur* n'a pas atteint sa limite de recherche et ne renvoie pas de limite.
Bad_DataLost	Il manque des données suite au démarrage/à l'arrêt/à la perte de la collecte.
Bad_DataUnavailable	Les données attendues ne sont pas disponibles pour l'intervalle de temps demandé en raison d'un volume non monté, d'un ensemble d'historiques hors ligne ou pour une raison similaire d'indisponibilité temporaire.
Bad_EntryExists	Les données ou *Événements* n'ont pas été correctement inséré(e)s car il existe une entrée correspondante.
Bad_NoEntryExists	Les données ou *Événements* n'ont pas été correctement mis(e)s à jour car il n'existe aucune entrée correspondante.
Bad_TimestampNotSupported	Le *Client* a demandé l'historique en utilisant un TimestampsToReturn que le *Serveur* ne prend pas en charge (c'est-à-dire le *ServerTimestamp* demandé lorsque le *Serveur* prend uniquement en charge *SourceTimestamp*)
Bad_InvalidArgument	Un ou plusieurs arguments ne sont pas valides ou sont absents.
Bad_AggregateListMismatch	La longueur de la liste des *Agrégats* est différente de celle de la liste des opérations.
Bad_AggregateConfigurationRejected	Le *Serveur* ne prend pas en charge l'*AggregateConfiguration* spécifiée pour le *Nœud*.
Bad_AggregateNotSupported	L'*Agrégat* spécifié n'est pas valide pour le *nœud* spécifié.
Bad_ArgumentsMissing	Voir l'IEC 62541-4:2015, Tableau 63, pour la description de ce code de résultat.
Bad_TypeDefinitionInvalid	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.
Bad_SourceNodeIdInvalid	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.
Bad_OutOfRange	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.
Bad_NotSupported	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.
Bad_IndexRangeInvalid	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.
Bad_NotWriteable	Voir l'IEC 62541-4:2015, Tableau 166, pour la description de ce code de résultat.

Tableau 17 – Codes de résultat de niveau d'opération Good

ID symbolique	Description
Good_NoData	Il n'existe aucune donnée pour l'intervalle de temps ou le filtre d'*Événements* demandé.
Good_EntryInserted	Les données ou *Événements* ont été correctement inséré(e)s dans la base de données historique.
Good_EntryReplaced	Les données ou *Événements* ont été correctement remplacé(e)s dans la base de données historique.
Good_DataIgnored	Le champ *Événement* a été ignoré et n'a pas été inséré dans la base de données historique.

Il peut être noté que les deux codes de statut Good et Bad apparaissent en l'absence de donnée ou lorsqu'il manque des données. En général, *Good_NoData* est utilisé lorsqu'aucune donnée n'a été trouvée lors de l'exécution d'une simple requête "Read". *Bad_NoData* est utilisé lorsque certaines actions sont demandées sur un intervalle et qu'aucune donnée ne peut être trouvée. La distinction existe si les utilisateurs tentent de réaliser une action sur un intervalle donné et qu'ils s'attendent à l'existence de données ou souhaitent être informés que l'action demandée ne peut pas être réalisée.

Good_NoData est renvoyé lorsque:

– ReadEvents, où *startTime=endTime,*
– les données ReadEvent sont demandées et qu'elles n'existent pas,
– ReadRaw, où les données sont demandées et n'existent pas.

Bad_NoData est renvoyé lorsque:

– les données ReadEvent sont demandées et que l'historique sous-jacent ne prend pas en charge le champ demandé,
– ReadProcessed, où les données sont demandées et n'existent pas,
– Toutes les requêtes Delete, où les données n'existent pas.

Les cas d'utilisation ci-dessus sont des exemples donnés à titre d'illustration. Des explications détaillées sur le renvoi de chaque code de statut sont données en 6.4 et 6.7

6.2.3 SemanticsChanged

Le *StatusCode* contient en outre un bit informationnel appelé *SemanticsChanged*. (Voir l'IEC 62541-4).

Il convient que les *Serveurs* UA qui mettent en place l'Accès à l'Historique OPC UA ne définissent pas ce bit, mais propagent plutôt le *StatusCode* qui a été stocké dans le référentiel de données. Il convient que le *Client* comprenne que le bit des valeurs de données renvoyées peut être défini.

6.3 Points de continuation

Le paramètre *continuationPoint* du *Service HistoryRead* est utilisé pour marquer le point à partir duquel se poursuit la lecture si toutes les valeurs n'ont pas pu être retournées dans une seule réponse. La valeur n'est pas connue du *Client* et elle est utilisée seulement pour maintenir les informations d'état pour le point de continuation pour le *Serveur*. Pour les requêtes *HistoricalDataNode*, un *Serveur* peut utiliser l'horodatage du dernier élément de données retourné si l'horodatage est unique. Cela réduit la nécessité de stocker des informations d'état dans le *Serveur* pour le point de continuation.

Le *Client* spécifie le nombre maximal de résultats par opération dans le *Message* de demande. Un *Serveur* ne doit pas en retourner plus que ce nombre de résultats, mais il peut

en retourner moins. Le *Serveur* attribue un *ContinuationPoint* s'il y a plus de résultats à retourner. Le *Serveur* peut renvoyer moins de résultats en raison de problèmes liés à la mémoire tampon ou à d'autres contraintes internes. Il peut également être exigé de renvoyer un *continuationPoint* en raison de contraintes liées au paramètre *HistoryRead*. Si le calcul d'un *Agrégat* prend du temps et que le délai d'expiration approche, le *Serveur* peut renvoyer des résultats partiels avec un point de continuation. Cela est possible si le calcul prend plus de temps que le délai d'expiration du *Client*. Dans certains cas, même le calcul d'un résultat d'*Agrégat* peut être plus long que le délai d'expiration du *Client*. Ensuite, le *Serveur* peut ne renvoyer aucun résultat avec un point de continuation permettant au *Serveur* de reprendre le calcul sur l'appel de lecture du *Client* suivant. Pour des discussions supplémentaires relatives aux *ContinuationPoints* et à l'*HistoryRead*, voir 6.4 relatif au paramètre *historyReadDetails* extensible individuel.

Si le *Client* spécifie un *ContinuationPoint*, les paramètres *HistoryReadDetails* et TimestampsToReturn sont ignorés, car la demande de paramètres différents n'a pas de sens en cas de continuation à partir d'un appel précédent. Il est admis de modifier le paramètre *dataEncoding* avec chaque requête.

Si le *Client* spécifie un *ContinuationPoint* qui ne correspond pas au dernier *ContinuationPoint* renvoyé du *Serveur*, le *Serveur* doit renvoyer une erreur *Bad_ContinuationPointInvalid*.

Si le paramètre *releaseContinuationPoints* est défini dans la requête, le *Serveur* ne doit pas renvoyer de données et doit libérer tous les *ContinuationPoints* transmis dans la requête. Si le *ContinuationPoint* d'une opération est absent ou non valide, le *StatusCode* de l'opération doit être *Bad_ContinuationPointInvalid.*

6.4 Paramètres HistoryReadDetails

6.4.1 Vue d'ensemble

Le *Service HistoryRead* défini dans l'IEC 62541-4 peut réaliser plusieurs fonctions différentes. Le paramètre *historyReadDetails* est un *Paramètre extensible* qui spécifie la fonction à réaliser et les détails spécifiques à cette fonction. Voir l'IEC 62541-4 pour la définition du *Paramètre extensible*. Le Tableau 18 répertorie les noms symboliques des structures de *Paramètre extensible* valides. Certaines structures exécutent des fonctions différentes selon les valeurs attribuées à ses paramètres associés. Pour plus de simplicité, une fonctionnalité de chaque structure est répertoriée. Par exemple, un texte tel que "utilisation de la fonctionnalité modifiée Read" fait référence à la fonction que le *Service HistoryRead* exécute à l'aide de la structure de *Paramètre extensible* ReadRawModifiedDetails, la valeur TRUE étant attribuée au paramètre booléen isReadModified.

Tableau 18 – HistoryReadDetails parameterTypeIds

Nom symbolique	Fonctionnalité	Description
ReadEventDetails	Read event	Cette structure sélectionne un ensemble d'*Événements* dans la base de données historique en spécifiant un filtre et un domaine temporel pour un ou plusieurs *Objets* ou *Vues*. Voir 6.4.2.1. Si ce paramètre est spécifié, le *Serveur* renvoie une structure *HistoryEvent* pour chaque opération (voir 6.5.4).
ReadRawModifiedDetails	Read raw	Cette structure sélectionne un ensemble de valeurs dans la base de données historique en spécifiant un domaine temporel pour une ou plusieurs *Variables*. Voir 6.4.3.1. Si ce paramètre est spécifié, le *Serveur* renvoie une structure *HistoryData* pour chaque opération (voir 6.5.2).
ReadRawModifiedDetails	Read modified	Ce paramètre sélectionne un ensemble de *valeurs modifiées* dans la base de données historique en spécifiant un domaine temporel pour une ou plusieurs *Variables*. Voir 6.4.3.1. Si ce paramètre est spécifié, le *Serveur* renvoie une structure *HistoryModifiedData* pour chaque opération (voir 6.5.3).
ReadProcessedDetails	Read processed	Cette structure sélectionne un ensemble de valeurs d'*Agrégat* dans la base de données historique en spécifiant un domaine temporel pour une ou plusieurs *Variables*. Voir 6.4.4.1. Si ce paramètre est spécifié, le *Serveur* renvoie une structure *HistoryData* pour l'opération (voir 6.5.2).
ReadAtTimeDetails	Read at time	Cette structure sélectionne un ensemble de valeurs brutes ou interpolées dans la base de données historique en spécifiant une série d'horodatages pour une ou plusieurs *Variables*. Voir 6.4.5.1. Si ce paramètre est spécifié, le *Serveur* renvoie une structure *HistoryData* pour chaque opération (voir 6.5.2).

6.4.2 Structure ReadEventDetails

6.4.2.1 Détails de la structure ReadEventDetails

Le Tableau 19 définit la structure *ReadEventDetails*. Ce paramètre est uniquement valide pour les *Objets* dont la valeur TRUE a été attribuée à l'*Attribut EventNotifier* (voir l'IEC 62541-3). Deux des trois paramètres (*numValuesPerNode*, *startTime* et *endTime*) doivent être spécifiés.

Tableau 19 – ReadEventDetails

Nom	Type	Description
ReadEventDetails	Structure	Spécifie les détails utilisés pour réaliser une lecture d'historique d'*Événements*
numValuesPerNode	Counter	Nombre maximal de valeurs renvoyées pour un *Nœud* sur l'intervalle de temps. Si une seule heure est spécifiée, l'intervalle de temps doit s'étendre pour renvoyer ce nombre de valeurs. La valeur par défaut 0 indique l'absence de valeur maximale.
startTime	UtcTime	Début de la période de lecture. La valeur par défaut de *DateTime.MinValue* indique que la *startTime* est Unspecified (non spécifiée).
endTime	UtcTime	Fin de la période de lecture. La valeur par défaut de *DateTime.MinValue* indique que l'*endTime* est Unspecified (non spécifiée).
Filtre	EventFilter	Filtre utilisé par le *Serveur* pour déterminer le *HistoricalEventNode* qu'il convient d'inclure. Ce paramètre doit être spécifié, et au moins un *EventField* est requis. Le type de paramètre *EventFilter* est extensible. Il est défini et utilisé de la même manière que lorsqu'il est défini pour les éléments de données surveillés spécifiés dans l'IEC 62541-4. Ce filtre spécifie également les *EventFields* qui sont à renvoyer dans le cadre de la requête.

6.4.2.2 Fonctionnalité ReadEvent

La structure *ReadEventDetails* est utilisée pour lire les *Événements* dans la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalEventNodes*. Les

Événements sont filtrés en fonction de la structure de filtre fournie. Ce filtre inclut les *EventFields* qui sont à renvoyer. Pour une description exhaustive du filtre, voir l'IEC 62541-4.

La *startTime* et l'*endTime* sont utilisées pour filtrer le champ Time pour les *Événements*.

Le domaine temporel de la requête est défini par *startTime*, *endTime* et *numValuesPerNode*. Au moins deux de ces valeurs doivent être spécifiées. Si *endTime* est inférieur à *startTime* ou si *endTime* et *numValuesPerNode* seuls sont spécifiés, les données sont renvoyées dans l'ordre inverse, les données les moins/plus récentes étant fournies en premier comme si le temps revenait en arrière. Si les trois éléments sont spécifiés, l'appel doit renvoyer jusqu'à *numValuesPerNode*, les résultats partant de *startTime* jusqu'à *endTime*, dans l'ordre croissant ou décroissant en fonction des valeurs relatives de *startTime* et *endTime*. Si *numValuesPerNode* est nul, toutes les valeurs de la plage sont renvoyées. La valeur par défaut est utilisée pour préciser quand *startTime*, *endTime* ou *numValuesPerNode* ne sont pas spécifiés.

Il est spécifiquement admis que *startTime* et *endTime* soient identiques. Cela permet au *Client* de demander l'*Événement* à un moment donné. Si *startTime* et *endTime* sont identiques, le temps est supposé avancer. En l'absence de données au moment spécifié, le *Serveur* doit renvoyer le *StatusCode Good_NoData*.

Si *startTime*, *endTime* et *numValuesPerNode* sont fournis, et si cet intervalle de temps contient plusieurs Événements *numValuesPerNode* pour un Nœud donné, seuls les Événements *numValuesPerNode* par Nœud sont renvoyés avec un *continuationPoint*. Si un *continuationPoint* est renvoyé, il convient qu'un *Client* qui souhaite obtenir les valeurs *numValuesPerNode* suivantes appelle de nouveau *HistoryRead* avec le *ContinuationPoint*.

Si un intervalle ne contient aucune donnée*, le StatusCode* correspondant doit être *Good_NoData*.

Le paramètre de *filtre* est utilisé pour déterminer les *Événements* historiques, et leurs champs correspondants sont renvoyés. Il est possible que les champs d'un *EventType* soient disponibles pour la mise à jour en temps réel, mais pas à partir de l'historique. Dans ce cas, une valeur de *StatusCode* est renvoyée pour un champ d'*Événement* qui ne peut pas être renvoyé. La valeur de *StatusCode* doit être *Bad_NoData*.

Si le *TimestampsToReturn* demandé n'est pas pris en charge pour un *Nœud*, l'opération doit renvoyer le *StatusCode Bad_TimestampNotSupported*. Lors de la lecture des *Événements*, cela s'applique uniquement aux champs d'*Événement* de type *DataValue*.

6.4.3 Structure ReadRawModifiedDetails

6.4.3.1 Détails de la structure ReadRawModifiedDetails

Le Tableau 20 définit la structure *ReadRawModifiedDetails*. Deux des trois paramètres (*numValuesPerNode*, *startTime* et *endTime*) doivent être spécifiés.

Tableau 20 – ReadRawModifiedDetails

Nom	Type	Description
ReadRawModifiedDetails	Structure	Spécifie les détails utilisés pour réaliser une lecture d'historique "Raw" ou "Modified".
isReadModified	Booléen	TRUE pour la fonctionnalité ReadModified, FALSE pour la fonctionnalité ReadRaw. La valeur par défaut est FALSE.
startTime	UtcTime	Début de la période de lecture. Attribuer la valeur par défaut de *DateTime.MinValue* si aucune heure de début particulière n'est spécifiée.
endTime	UtcTime	Fin de la période de lecture. Attribuer la valeur par défaut de *DateTime.MinValue* si aucune heure de fin particulière n'est spécifiée.
numValuesPerNode	Counter	Nombre maximal de valeurs renvoyées pour un *Nœud* sur l'intervalle de temps. Si une seule heure est spécifiée, l'intervalle de temps doit s'étendre pour renvoyer ce nombre de valeurs. La valeur par défaut 0 indique l'absence de valeur maximale.
returnBounds	Booléen	Paramètre booléen avec les valeurs suivantes: TRUE Il convient de renvoyer les Valeurs limites FALSE Tous les autres cas

6.4.3.2 Fonctionnalité ReadRaw

Si la structure est utilisée pour la lecture des Valeurs *Brutes* (la valeur FALSE est attribuée à isReadModified), les valeurs, qualités et horodatages sont lus dans la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*. Ce paramètre est censé être utilisé par un *Client* qui souhaite sauvegarder les données réelles dans l'historique. Les données réelles peuvent être compressées. Il peut également s'agir de données brutes collectées pour l'élément en fonction de l'historique et des règles de stockage appelées lors de la sauvegarde des valeurs de l'élément. Si returnBounds est TRUE, les Valeurs limites du domaine temporel sont renvoyées. Les Valeurs limites facultatives sont fournies pour permettre au *Client* d'interpoler les valeurs des heures de début et de fin lors de l'établissement de la tendance des données réelles sur un écran.

Le domaine temporel de la requête est défini par *startTime*, *endTime* et *numValuesPerNode*. Au moins deux de ces valeurs doivent être spécifiées. Si *endTime* est inférieur à *startTime* ou si *endTime* et *numValuesPerNode* seuls sont spécifiés, les données sont renvoyées dans l'ordre inverse, les données les plus anciennes venant en premier comme si le temps revenait en arrière. Si les trois éléments sont spécifiés, l'appel doit renvoyer jusqu'à *numValuesPerNode*, les résultats partant de *startTime* jusqu'à *endTime*, dans l'ordre croissant ou décroissant en fonction des valeurs relatives de *startTime* et *endTime*. Si *numValuesPerNode* est nul, toutes les valeurs de la plage sont renvoyées. Une valeur par défaut de *DateTime.MinValue* (voir l'IEC 62541-6) est utilisée pour préciser quand *startTime* ou *endTime* n'est pas spécifiée.

Il est spécifiquement admis que *startTime* et *endTime* soient identiques. Cela permet au *Client* de demander juste une valeur. Si *startTime* et *endTime* sont identiques, le temps est supposé avancer. Il n'est pas admis que le *Serveur* renvoie un *StatusCode Bad_InvalidArgument* si le domaine temporel demandé est hors de la plage du *Serveur*. Ces cas particuliers doivent être traités comme un intervalle ne contenant aucune donnée.

Si *startTime*, *endTime* et *numValuesPerNode* sont fournis, et si cet intervalle de temps contient plusieurs valeurs *numValuesPerNode* pour un *Nœud* donné, seules les valeurs *numValuesPerNode* par *Nœud* sont renvoyées avec un *continuationPoint*. Si un *continuationPoint* est renvoyé, il convient qu'un *Client* qui souhaite obtenir les valeurs *numValuesPerNode* suivantes appelle de nouveau ReadRaw avec le *continuationPoint* défini.

Si des Valeurs limites sont demandées et qu'une valeur *numValuesPerNode* différente de zéro a été spécifiée, les Valeurs limites renvoyées sont incluses dans le comptage *numValuesPerNode*. Si la valeur de *numValuesPerNode* est 1, seule la limite de début est

renvoyée (la limite de fin est renvoyée si l'ordre inverse est nécessaire). Si la valeur de *numValuesPerNode* est 2, la limite de début et le premier point de données sont renvoyés (la limite de fin est renvoyée si l'ordre inverse est nécessaire). Si des Valeurs limites sont demandées et qu'aucune n'a été trouvée, l'entrée du *StatusCode* correspondant est définie à *Bad_BoundNotFound*, un horodatage égal à l'heure de début ou de fin selon le cas, et une valeur nulle. Le degré de recherche des Valeurs limites en arrière ou en avant dans l'historique dépend du *Serveur*.

Pour un intervalle ne contenant aucune donnée, si des Valeurs limites ne sont pas demandées, le *StatusCode* correspondant doit être *Good_NoData*. Si des Valeurs limites sont demandées et qu'une ou les deux existent, le code de résultat renvoyé est Success et la ou les Valeurs limites sont renvoyées.

En présence de plusieurs valeurs pour un horodatage donné, toutes les valeurs sont considérées comme étant des *Valeurs modifiées*, sauf les plus récentes, et le *Serveur* doit renvoyer la valeur la plus récente. Si le *Serveur* renvoie une valeur qui masque d'autres valeurs à un horodatage donné, il doit définir le bit *ExtraData* dans le *StatusCode* associé à cette valeur. Si le Serveur contient des informations complémentaires relatives à une valeur, le bit ExtraData doit également être défini. Ce bit indique que les ModifiedValues sont disponibles pour l'extraction (voir 6.4.3.3).

Si le TimestampsToReturn demandé n'est pas pris en charge pour un *Nœud*, l'opération doit renvoyer le *StatusCode Bad_TimestampNotSupported*.

6.4.3.3 Fonctionnalité ReadModified

Si cette structure est utilisée pour la lecture des Valeurs *Modifiées* (isReadModified est TRUE), elle lit les valeurs, le *StatusCodes*, les horodatages, le type de modification, l'ID utilisateur et l'horodatage de la modification dans la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*. Si plusieurs valeurs sont remplacées, le *Serveur* doit toutes les renvoyer. L'updateType spécifie la valeur renvoyée dans l'enregistrement de modification. Si updateType est INSERT, la valeur est la nouvelle valeur qui a été insérée. Si updateType est une autre valeur, la valeur est l'ancienne valeur qui a été modifiée.

Cette fonction a pour objet de lire les valeurs dans l'historique qui ont été *Modifiées.* La valeur FALSE doit être attribuée au paramètre returnBounds dans ce cas, sinon le *Serveur* renvoie un *StatusCode Bad_InvalidArgument*.

Le domaine de la requête est défini par *startTime*, *endTime* et *numValuesPerNode*. Au moins deux de ces valeurs doivent être spécifiées. Si *endTime* est inférieure à *startTime* ou si *endTime* et *numValuesPerNode* seuls sont spécifiés, les données doivent être renvoyées dans l'ordre inverse, les données les plus anciennes venant en premier. Si les trois éléments sont spécifiés, l'appel doit renvoyer jusqu'à *numValuesPerNode*, les résultats partant de *StartTime* jusqu'à *EndTime*, dans l'ordre croissant ou décroissant en fonction des valeurs relatives de *StartTime* et *EndTime*. Si cet intervalle de temps contient plusieurs *numValuesPerNode* pour un *Nœud* donné, seules les valeurs *numValuesPerNode* par *Nœud* sont renvoyées avec un *continuationPoint*. Si un *continuationPoint* est renvoyé, il convient qu'un *Client* qui souhaite obtenir les valeurs *numValuesPerNode* suivantes appelle de nouveau ReadRaw avec le *continuationPoint* défini. Si *numValuesPerNode* est nul, toutes les valeurs de la plage sont renvoyées. Si le Serveur ne peut pas renvoyer toutes les *valeurs modifiées* pour un horodatage donné, il doit renvoyer les valeurs modifiées avec le même horodatage dans des appels subséquents.

Si une valeur a été modifiée plusieurs fois, toutes les valeurs correspondant à l'horodatage sont renvoyées. Cela signifie qu'un horodatage peut apparaître plusieurs fois dans la matrice. Il convient que les valeurs renvoyées avec le même horodatage soient ordonnées de l'horodatage de la modification la plus récente vers celui de la plus ancienne, si *startTime* est inférieure ou égale à *endTime*. Si *endTime* est inférieure à *startTime*, les valeurs renvoyées sont ordonnées de l'horodatage de la modification la plus ancienne vers celui de la plus

récente. Le maintien de plusieurs modifications ou celui de la modification la plus récente dépend du *Serveur*.

Un *Serveur* n'a pas à créer un enregistrement de modification des données s'il a été en premier lieu ajouté à l'ensemble d'historiques. Si cela n'est pas le cas, il doit définir le bit ExtraData et le *Client* peut lire l'enregistrement de modification à l'aide d'un appel ReadModified. Si les données sont par la suite modifiées, le *Serveur* doit créer un deuxième enregistrement de modification renvoyé avec l'enregistrement de modification d'origine à chaque fois qu'un *Client* utilise l'appel ReadModified, si le Serveur prend en charge plusieurs enregistrements de modification par horodatage.

Si le TimestampsToReturn demandé n'est pas pris en charge pour un *Nœud*, l'opération doit renvoyer le *StatusCode Bad_TimestampNotSupported*.

6.4.4 Structure ReadProcessedDetails

6.4.4.1 Détails de la structure ReadProcessedDetails

Le Tableau 21 définit la structure de ReadProcessedDetails.

Tableau 21 – ReadProcessedDetails

Nom	Type	Description
ReadProcessedDetails	Structure	Spécifie les détails utilisés pour réaliser une lecture d'historique "traitée".
startTime	UtcTime	Début de la période de lecture.
endTime	UtcTime	Fin de la période de lecture.
ProcessingInterval	Durée	Intervalle entre les valeurs d'*Agrégat* renvoyées. La valeur 0 indique qu'aucun *ProcessingInterval* n'est défini.
aggregateType[]	NodeId	Le *NodeId* de l'objet HistoryAggregate qui indique la liste des *Agrégats* à utiliser pour récupérer l'historique traité. Voir l'IEC 62541-13 pour plus de détails.
aggregateConfiguration	Configuration d'Agrégat	Structure de configuration d'*Agrégat.*
useSeverCapabilitiesDefaults	Booléen	Voir l'IEC 62541-4.
TreatUncertainAsBad	Booléen	Voir l'IEC 62541-13.
PercentDataBad	UInt8	Voir l'IEC 62541-13.
PercentDataGood	UInt8	Voir l'IEC 62541-13.
UseSlopedExtrapolation	Booléen	Voir l'IEC 62541-13.

Voir l'IEC 62541-13 pour plus de détails sur les valeurs possibles de *NodeId* pour le paramètre *HistoryAggregateType*.

6.4.4.2 Fonctionnalité traitée Read

Cette structure permet de calculer les valeurs d'*Agrégat*, les qualités et les horodatages à partir de données de la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*. Le domaine temporel est divisé en intervalles de durée *ProcessingInterval*. Le type d'*Agrégat* spécifié est calculé pour chaque intervalle, en commençant par *startTime* en utilisant les données du *ProcessingInterval* suivant.

Par exemple, cette fonction peut fournir des statistiques horaires (Maximum, Minimum et Average, par exemple) pour chaque élément au cours du domaine temporel spécifié lorsque le *ProcessingInterval* est de 1 h.

Le domaine de la requête est défini par *startTime*, *endTime* et *ProcessingInterval*. Les trois doivent être spécifiés. Si *endTime* est inférieure à *startTime*, les données doivent être renvoyées dans l'ordre inverse, les données les plus anciennes venant en premier. Si *startTime* et *endTime* sont identiques, le *Serveur* doit renvoyer *Bad_InvalidArgument*, puisqu'aucun moyen ne permet d'interpréter ce type de cas.

Le paramètre aggregateType[] permet à un *Client* de demander plusieurs calculs d'*Agrégat* par NodeId demandé. Si plusieurs *Agrégats* sont demandés, un nombre correspondant d'entrées est requis dans la matrice *NodesToRead*.

Par exemple, pour demander l'*Agrégat* Min pour *NodeId* FIC101, FIC102 et les *Agrégats* Min et Max pour *NodeId,* FIC103 exige que *NodeId* FIC103 apparaisse deux fois dans le paramètre de requête de la matrice *NodesToRead*.

aggregateType[]	*NodesToRead*[]
Min	FIC101
Min	FIC102
Min	FIC103
Max	FIC103

Si la matrice des *Agrégats* ne correspond pas à celle de *NodesToRead*, le Serveur doit renvoyer un *StatusCode* de *Bad_AggregateListMismatch.*

Le paramètre aggregateConfiguration permet à un *Client* de passer outre les paramètres de configuration de l'*Agrégat* fournis par l'*Objet* AggregateConfiguration, par appel. Voir l'IEC 62541-13 pour plus d'informations sur les configurations d'*Agrégat*. Si le Serveur ne prend pas en charge la possibilité de passer outre les paramètres de configuration d'*Agrégat*, il doit renvoyer un *StatusCode* de *Bad_ AggregateConfigurationRejected.* Si l'*Agrégat* n'est pas valide pour le *Nœud*, le *StatusCode* doit être *Bad_AggregateNotSupported.*

Les valeurs utilisées dans le calcul de l'*Agrégat* pour chaque intervalle doivent correspondre exactement à l'horodatage au début de l'intervalle, mais ne doivent pas inclure les valeurs tombant directement sur l'horodatage de fin de l'intervalle. Par conséquent, chaque valeur doit être incluse une seule fois dans le calcul. Si le domaine temporel est dans l'ordre inverse, le dernier horodatage est considéré comme étant celui qui commence le sous-intervalle, le premier horodatage étant celui qui le termine. Noter que cela signifie que la simple permutation des heures de début et de fin ne permet pas de revenir aux mêmes valeurs dans l'ordre inverse, les intervalles demandés dans les deux cas étant différents.

Si le calcul d'un *Agrégat* prend du temps, le *Serveur* peut renvoyer des résultats partiels avec un point de continuation. Cela est possible si le calcul prend plus de temps que l'indication de délai d'expiration du *Client*. Dans certains cas, même le calcul d'un résultat d'*Agrégat* peut être plus long que l'indication de délai d'expiration du *Client*. Ensuite, le *Serveur* peut ne renvoyer aucun résultat avec un point de continuation permettant au *Serveur* de reprendre le calcul sur l'appel de lecture du *Client* suivant.

Voir l'IEC 62541-13 pour la gestion des cas spécifiques à l'*Agrégat*.

6.4.5 Structure ReadAtTimeDetails

6.4.5.1 Détails de la structure ReadAtTimeDetails

Le Tableau 22 définit la structure ReadAtTimeDetails.

Tableau 22 – ReadAtTimeDetails

Nom	Type	Description
ReadAtTimeDetails	Structure	Spécifie les détails utilisés pour réaliser une lecture d'historique "à temps".
reqTimes []	UtcTime	Les entrées définissent les horodatages spécifiques des valeurs à lire.
useSimpleBounds	Booléen	Utiliser SimpleBounds pour déterminer la valeur à l'horodatage spécifique.

6.4.5.2 Fonctionnalité ReadAtTime

La structure ReadAtTimeDetails permet de lire les valeurs et qualités dans la base de données historique pour les horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. Cette fonction est censée fournir des valeurs pour la corrélation avec d'autres valeurs d'un horodatage connu. Par exemple, un *Client* peut souhaiter lire les valeurs de capteurs lors de la collecte des échantillons de laboratoire.

L'ordre des valeurs et des qualités renvoyées doit correspondre à celui des horodatages fournis dans la requête.

En l'absence de valeur pour un horodatage spécifié, une valeur doit être *Interpolée* à partir des valeurs environnantes afin de représenter la valeur à l'horodatage spécifié. L'interpolation suit les mêmes règles que l'*Agrégat Interpolated* présenté dans l'IEC 62541-13.

Si le fanion useSimpleBounds est True et que l'Interpolation est exigée, *SimpleBounds* est utilisé pour calculer la valeur de données.

Si une valeur est trouvée pour l'horodatage spécifié, le *Serveur* attribue la valeur *Raw* au *StatusCode InfoBits*. Si la valeur est *Interpolée* à partir de valeurs environnantes, le *Serveur* attribue la valeur *Interpolated* au *StatusCode InfoBits*.

Si le TimestampsToReturn demandé n'est pas pris en charge pour un *Nœud*, l'opération doit renvoyer le *StatusCode Bad_TimestampNotSupported*.

6.5 Paramètres HistoryData renvoyés

6.5.1 Vue d'ensemble

Le *Service HistoryRead* renvoie différents types de données, selon que la requête concerne l'*Attribut* Valeur d'un *Nœud* ou les *Événements* historiques d'un *Nœud*. L'historyData est un *Paramètre extensible* dont la structure dépend des fonctions à réaliser pour le paramètre *HistoryReadDetails*. Voir l'IEC 62541-4 pour plus de détails sur les *Paramètres extensibles*.

6.5.2 Type HistoryData

Le Tableau 23 définit la structure de l'*HistoryData* utilisé pour les données à renvoyer dans un *HistoryRead*.

Tableau 23 – Détails d'HistoryData

Nom	Type	Description
dataValues[]	DataValue	Matrice de valeurs de données historiques pour le *Nœud*. La taille de la matrice dépend des paramètres de données demandés.

6.5.3 Type HistoryModifiedData

Le Tableau 24 définit la structure de l'*HistoryModifiedData* utilisé pour les données à renvoyer dans un *HistoryRead* lorsque IsReadModified = True.

Tableau 24 – Détails d'HistoryModifiedData

Nom	Type	Description
dataValues[]	DataValue	Matrice de valeurs de données historiques pour le *Nœud*. La taille de la matrice dépend des paramètres de données demandés.
modificationInfos[]	ModificationInfo	
Username	Chaîne	Nom de l'utilisateur qui a apporté la modification. La prise en charge de ce champ est facultative. Une valeur nulle doit être renvoyée si elle n'est pas définie.
modificationTime	UtcTime	Heure à laquelle la modification a été apportée. La prise en charge de ce champ est facultative. Une valeur nulle doit être renvoyée si elle n'est pas définie.
updateType	HistoryUpdateType	Type de modification de l'élément.

6.5.4 Type HistoryEvent

Le Tableau 25 définit le paramètre HistoryEvent utilisé pour les lectures d'*Événement* Historique.

HistoryEvent définit une structure de tableau utilisée pour renvoyer les champs d'*Événement* à un *Historical Read*. La structure se présente sous la forme d'un tableau composé d'un ou de plusieurs *Événements*, contenant chacun une matrice d'un ou de plusieurs champs. Le choix et l'ordre des champs renvoyés pour chaque *Événement* sont identiques à ceux du paramètre sélectionné de l'*EventFilter*.

Tableau 25 – Détails d'HistoryEvent

Nom	Type	Description
Events []	HistoryEventField List	Liste des *Événements* distribués.
eventFields []	BaseDataType	Liste des champs d'*Événement* sélectionnés. Il s'agira d'une correspondance univoque avec les champs sélectionnés dans l'*EventFilter*.

6.6 Énumération HistoryUpdateType

Le Tableau 26 définit l'énumération *HistoryUpdate*.

Tableau 26 – Énumération HistoryUpdateType

Nom	Description
INSERT_1	Les données ont été insérées.
REPLACE_2	Les données ont été remplacées.
UPDATE_3	Les données ont été insérées ou remplacées.
DELETE_4	Les données ont été supprimées.

6.7 Énumération PerformUpdateType

Le Tableau 27 définit l'énumération PerformUpdateType.

Tableau 27 – Énumération PerformUpdateType

Nom	Description
INSERT_1	Les données ont été insérées.
REPLACE_2	Les données ont été remplacées.
UPDATE_3	Les données ont été insérées ou remplacées.
DELETE_4	Les données ont été supprimées.

6.8 Paramètre HistoryUpdateDetails

6.8.1 Vue d'ensemble

Le *Service HistoryUpdate* défini dans l'IEC 62541-4 peut réaliser plusieurs fonctions différentes. Le paramètre *historyUpdateDetails* est un *Paramètre extensible* qui spécifie la fonction à réaliser et les détails spécifiques à cette fonction. Voir l'IEC 62541-4 pour la définition du *Paramètre extensible*. Le Tableau 28 répertorie les noms symboliques des structures de *Paramètre extensible* valides. Certaines structures exécutent des fonctions différentes selon les valeurs attribuées à ses paramètres associés. Pour plus de simplicité, une fonctionnalité de chaque structure est répertoriée. Par exemple, un texte tel que "utilisation de la fonctionnalité de remplacement des données" fait référence à la fonction que le *Service HistoryUpdate* exécute à l'aide de la structure de *Paramètre extensible UpdateDataDetails*, la valeur REPLACE_2 étant attribuée au paramètre d'énumération performInsertReplace.

Tableau 28 – TypeIds du paramètre HistoryUpdateDetails

Nom symbolique	Fonctionnalité	Description
UpdateDataDetails	Insertion de données	Cette fonction permet d'insérer des données dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateDataDetails	Remplacement de données	Cette fonction remplace les valeurs existantes dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateDataDetails	Mise à jour de données	Cette fonction permet d'insérer ou de remplacer des valeurs dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateStructureDataDetails	Insertion de données	Cette fonction permet d'insérer de nouvelles *Données Historiques Structurées* ou *Annotations* dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateStructureDataDetails	Remplacement de données	Cette fonction permet de remplacer des *Données Historiques Structurées* ou *Annotations* dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateStructureDataDetails	Mise à jour de données	Cette fonction permet d'insérer ou de remplacer des *Données Historiques Structurées* ou *Annotations* dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.

Nom symbolique	Fonctionnalité	Description
UpdateStructureDataDetails	Suppression de données	Cette fonction permet de supprimer des *Données Historiques Structurées* ou *Annotations* de la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. La valeur de la *Variable* est représentée par une valeur composite définie par le type de données *DataValue*.
UpdateEventDetails	Insertion d'événements	Cette fonction permet d'insérer de nouveaux *Événements* dans la base de données historique d'un ou de plusieurs *HistoricalEventNodes*.
UpdateEventDetails	Remplacement d'événements	Cette fonction permet de remplacer les valeurs des champs dans les *Événements* existants dans la base de données historique d'un ou de plusieurs *HistoricalEventNodes*.
UpdateEventDetails	Mise à jour d'événements	Cette fonction permet d'insérer de nouveaux *Événements* ou de remplacer les *Événements* existants dans la base de données historique d'un ou de plusieurs *HistoricalEventNodes*.
DeleteRawModifiedDetails	Suppression de valeur brute	Cette fonction permet de supprimer toutes les valeurs de la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*.
DeleteRawModifiedDetails	DeleteModified	Certains historiques peuvent stocker plusieurs valeurs au même horodatage. Cette fonction supprime les valeurs et qualités spécifiées pour l'horodatage spécifié d'un ou de plusieurs *HistoricalDataNodes*.
DeleteAtTimeDetails	DeleteAtTime	Cette fonction permet de supprimer toutes les valeurs de la base de données historique pour les horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*.
DeleteEventDetails	Suppression d'événement	Cette fonction permet de supprimer les *Événements* de la base de données historique pour le filtre spécifié d'un ou de plusieurs *HistoricalEventNodes*.

Le *Service HistoryUpdate* est utilisé pour mettre à jour ou supprimer les *DataValues* et les *Événements*. Pour des raisons de simplicité, le terme "entrée" est utilisé pour indiquer *DataValue* ou *Événement*, selon le contexte dans lequel il est utilisé. Les exigences d'audit pour les *Services* History sont décrites dans l'IEC 62541-4. Cette description part du principe que l'utilisateur qui formule la requête et que le *Serveur* qui la traite prennent en charge la capacité de mise à jour des entrées. Voir l'IEC 62541-3 pour une description des *Attributs* qui présentent la prise en charge des Mises à jour d'Historique.

6.8.2 Structure UpdateDataDetails

6.8.2.1 Détails de la structure UpdateDataDetails

Le Tableau 29 définit la structure UpdateDataDetails.

Tableau 29 – UpdateDataDetails

<table>
<tr><th>Nom</th><th>Type</th><th>Description</th></tr>
<tr><td>UpdateDataDetails</td><td>Structure</td><td>Détails de l'insertion, du remplacement et de l'insertion/remplacement des mises à jour d'historique.</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>ID de nœud de l'Objet à mettre à jour.</td></tr>
<tr><td>performInsertReplace</td><td>PerformUpdate Type</td><td>Valeur qui détermine l'action d'insertion, de remplacement ou de mise à jour qui est réalisée.
<table>
<tr><th>Valeur</th><th>Description</th></tr>
<tr><td>INSERT_1</td><td>Voir 6.8.2.2.</td></tr>
<tr><td>REPLACE_2</td><td>Voir 6.8.2.3.</td></tr>
<tr><td>UPDATE_3</td><td>Voir 6.8.2.4.</td></tr>
</table></td></tr>
<tr><td>updateValues[]</td><td>DataValue</td><td>Nouvelles valeurs à insérer ou remplacer.</td></tr>
</table>

6.8.2.2 Fonctionnalité d'insertion de données

Le paramétrage performInsertReplace = INSERT_1 permet d'insérer les entrées dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. Si une entrée existe à l'horodatage spécifié, la nouvelle entrée ne doit pas être insérée, le *StatusCode* doit plutôt indiquer *Bad_EntryExists*.

Cette fonction permet d'insérer de nouvelles entrées aux horodatages spécifiés (l'insertion de données de laboratoire pour refléter l'heure de la collecte de données, par exemple).

6.8.2.3 Fonctionnalité de remplacement de données

Le paramétrage performInsertReplace = REPLACE_2 permet de remplacer les entrées dans la base de données historique aux horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. Si aucune entrée n'existe à l'horodatage spécifié, la nouvelle entrée ne doit pas être insérée, sinon le *StatusCode* doit indiquer *Bad_NoEntryExists*.

Cette fonction permet de remplacer les entrées existantes à l'horodatage spécifié (des données de laboratoire correctes ayant été traitées de manière incorrecte et insérées dans la base de données historique, par exemple).

6.8.2.4 Fonctionnalité de mise à jour de données

Le paramétrage performInsertReplace = UPDATE_3 permet d'insérer ou de remplacer des entrées dans la base de données historique pour les horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*. Si l'élément comporte une entrée à l'horodatage spécifié, la nouvelle entrée remplace l'ancienne. En l'absence d'entrée à cet horodatage, la fonction insère les nouvelles données.

Un *Serveur* peut créer une *valeur modifiée* pour une valeur remplacée ou insérée (voir 3.1.6). Toutefois, cela n'est pas requis.

Cette fonction est censée insérer/remplacer sans condition des valeurs et qualités (correction de valeurs de capteurs incorrects, par exemple).

Good en tant que *StatusCode* pour une entrée individuelle est admis lorsque le *Serveur* n'est pas en mesure de dire si une valeur était déjà présente à cet horodatage. Si le *Serveur* peut déterminer si la nouvelle entrée remplace une entrée déjà présente, il convient qu'il utilise *Good_EntryInserted* ou *Good_EntryReplaced* pour renvoyer cette information.

6.8.3 Structure UpdateStructureDataDetails

6.8.3.1 Détails de la structure UpdateStructureDataDetails

Le Tableau 30 définit la structure UpdateStructureDataDetails.

Tableau 30 – UpdateStructureDataDetails

<table>
<tr><th>Nom</th><th>Type</th><th>Description</th></tr>
<tr><td>UpdateStructureDataDetails</td><td>Structure</td><td>Détails des mises à jour de données historiques.</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>ID de nœud de l'Objet à mettre à jour.</td></tr>
<tr><td>performInsertReplace</td><td>PerformUpdate Type</td><td>Valeur qui détermine l'action d'insertion, de remplacement ou de mise à jour qui est réalisée.
<table>
<tr><th>Valeur</th><th>Description</th></tr>
<tr><td>INSERT_1</td><td>Voir 6.8.3.3.</td></tr>
<tr><td>REPLACE_2</td><td>Voir 6.8.3.4.</td></tr>
<tr><td>UPDATE_3</td><td>Voir 6.8.3.5.</td></tr>
<tr><td>REMOVE_4</td><td>Voir 6.8.3.6.</td></tr>
</table></td></tr>
<tr><td>updateValue[]</td><td>DataValue</td><td>Nouvelles valeurs à insérer, remplacer ou supprimer.</td></tr>
</table>

6.8.3.2 Caractère unique spécifié des Données Historiques Structurées

Les *Données Historiques Structurées* fournissent les métadonnées décrivant une entrée dans la base de données historique. Le *Serveur* doit définir ce que signifie le caractère unique de chaque type de structure de *Données Historiques Structurées*. Par exemple, un *Serveur* peut uniquement permettre une *Annotation* par horodatage, ce qui signifie que l'horodatage est la clé unique de la structure. Un autre *Serveur* peut permettre plusieurs *Annotations* par utilisateur. Une combinaison nom d'utilisateur/horodatage/message peut faire office de clé unique pour la structure. En 6.8.3.3,6.8.3.4,6.8.3.5 et 6.8.3.6, les expressions "les *Données Historiques Structurées* existent" et "aux paramètres spécifiés" signifient qu'une entrée correspondante a été trouvée à l'horodatage spécifié à l'aide des critères de caractère unique du *Serveur*.

Si le Client souhaite remplacer un paramètre faisant partie des critères de caractère unique, le StatusCode obtenu est *Bad_NoEntryExists*. Le Client doit supprimer la structure existante et insérer la nouvelle structure.

6.8.3.3 Fonctionnalité d'insertion

Le paramétrage performInsertReplace = INSERT_1 permet d'insérer les *Données Historiques Structurées* (les *Annotations,* par exemple) dans la base de données historique aux paramètres spécifiés d'une ou de plusieurs *Propriétés* des *HistoricalDataNodes.*

Si une entrée de *Données Historiques Structurées* existe déjà aux paramètres spécifiés, le *StatusCode* doit indiquer *Bad_EntryExists*.

6.8.3.4 Fonctionnalité de remplacement

Le paramétrage performInsertReplace = REPLACE_2 permet de remplacer les *Données Historiques Structurées* (les *Annotations,* par exemple) dans la base de données historique aux paramètres spécifiés d'une ou de plusieurs *Propriétés* des *HistoricalDataNodes.*

Si une entrée de *Données Historiques Structurées* n'existe pas déjà aux paramètres spécifiés, le *StatusCode* doit indiquer *Bad_NoEntryExists*.

6.8.3.5 Fonctionnalité de mise à jour

Le paramétrage performInsertReplace = UPDATE_3 permet d'insérer ou de remplacer les *Données Historiques Structurées* (les *Annotations,* par exemple) dans la base de données historique aux paramètres spécifiés d'une ou de plusieurs *Propriétés* des *HistoricalDataNodes.*

Si une entrée de *Données Historiques Structurées* existe déjà aux paramètres spécifiés, elle est supprimée et la valeur fournie par le *Client* est insérée. S'il n'existe aucune donnée, la nouvelle entrée est insérée.

Si une entrée existante a été remplacée avec succès, le *StatusCode* doit être *Good_EntryReplaced.* Si une entrée a été créée, le *StatusCode* doit être *Good_EntryInserted.* Si le *Serveur* ne peut pas déterminer s'il a remplacé ou inséré une entrée, le *StatusCode* doit être *Good.*

6.8.3.6 Fonctionnalité de suppression

Le paramétrage performInsertReplace = REMOVE_4 permet de supprimer les *Données Historiques Structurées* (les *Annotations,* par exemple) de la base de données historique aux paramètres spécifiés d'une ou de plusieurs *Propriétés* des *HistoricalDataNodes.*

S'il existe une entrée de *Données Historiques structurées* aux paramètres spécifiés, elle est supprimée. Si les *Données Historiques Structurées* n'existent pas déjà aux paramètres spécifiés, le *StatusCode* doit indiquer *Bad_NoEntryExists*.

6.8.4 Structure UpdateEventDetails

6.8.4.1 Détails de la structure UpdateEventDetails

Le Tableau 31 définit la structure UpdateEventDetails.

Tableau 31 – UpdateEventDetails

<table>
<tr><th>Nom</th><th>Type</th><th>Description</th></tr>
<tr><td>UpdateEventDetails</td><td>Structure</td><td>Détails de l'insertion, du remplacement et de l'insertion/remplacement des mises à jour d'Événement historique.</td></tr>
<tr><td>nodeId</td><td>NodeId</td><td>ID de nœud de l'Objet à mettre à jour.</td></tr>
<tr><td>performInsertReplace</td><td>PerformUpdateType</td><td>Valeur qui détermine l'action d'insertion, de remplacement ou de mise à jour qui est réalisée.
<table>
<tr><th>Valeur</th><th>Description</th></tr>
<tr><td>INSERT_1</td><td>Procéder à l'insertion d'Événement (voir 6.8.4.2).</td></tr>
<tr><td>REPLACE_2</td><td>Procéder au remplacement d'Événement (voir 6.8.4.3).</td></tr>
<tr><td>UPDATE_3</td><td>Procéder à la mise à jour d'Événement (voir 6.8.4.4).</td></tr>
</table></td></tr>
<tr><td>filtre</td><td>EventFilter</td><td>Si l'historique de la Notification est conforme à l'EventFilter, l'historique de la Notification est mis à jour.</td></tr>
<tr><td>eventData[]</td><td>HistoryEventFieldList</td><td>Liste des Notifications d'Événement à insérer ou mettre à jour (voir 6.5.4 pour la définition d'HistoryEventFieldList).</td></tr>
</table>

6.8.4.2 Fonctionnalité d'insertion d'événement

Cette fonction est censée insérer de nouvelles entrées (remplissage des *Événements* historiques, par exemple).

Le paramétrage performInsertReplace = INSERT_1 permet d'insérer des entrées dans la base de données historique d'*Événement* d'un ou de plusieurs *HistoricalEventNodes*. Le paramètre *whereClause* de l'*EventFilter* doit être vide. Le *SelectClause* doit au moins fournir les champs d'Événement suivants: *EventType* et Time. Il est également recommandé de fournir les champs *SourceNode* et *SourceName*. Si l'un des champs requis n'est pas fourni, le *statusCode* doit indiquer *Bad_ArgumentsMissing*. Si l'historique ne prend pas en charge l'archivage de l'*EventType* spécifié, le *statusCode* doit indiquer *Bad_TypeDefinitionInvalid*. Si le *SourceNode* n'est pas la source valide des *Événements*, l'*entrée operationResults* connexe doit indiquer *Bad_SourceNodeIdInvalid.* Si *Time* ne tombe pas dans la plage qui peut être

stockée, l'*entrée operationResults* connexe doit indiquer *Bad_OutOfRange.* Si la *selectClause* ne contient pas les champs obligatoires pour l'*EventType,* le *statusCode* doit indiquer *Bad_ArgumentsMissing.* Si la *selectClause* spécifie des champs qui ne sont pas valides pour l'*EventType* ou qui ne peuvent pas être sauvegardés par l'historique, l'*entrée operationResults* connexe doit indiquer *Good_DataIgnored*. Des informations supplémentaires relatives aux champs ignorés doivent être fournies par l'intermédiaire du *DiagnosticInformation* lié aux *operationResults*. Le symbolicId contient l'indice de chaque champ ignoré séparé par un espace, le localizedText contenant les noms symboliques des champs ignorés.

L'*EventId* est une valeur opaque générée par le *Serveur* et un *Client* ne peut pas supposer qu'il sait comment créer des *EventIds* valides. Un Serveur doit être en mesure de générer une valeur par défaut appropriée pour le champ *EventId*. Si un *Client* spécifie l'*EventId* dans la *selectClause* et qu'il correspond à un *Événement* existant, le *statusCode* doit indiquer *Bad_EntryExists*. Un *Client* doit utiliser *HistoryRead* pour reconnaître les *EventIds* générés automatiquement

Si une erreur se produit lors du traitement des champs individuels, l'*entrée operationResults* connexe doit indiquer *Bad_InvalidArgument*, et les champs non valides doivent être indiqués dans le *DiagnosticInformation* lié à l'*entrée operationResults*.

Le paramètre *IndexRange* de *SimpleAttributeOperand* n'est pas valide pour les opérations d'insertion, et le *StatusCode* doit spécifier Bad_*IndexRangeInvalid* si ce paramètre est spécifié.

Un *Client* peut demander au Serveur de choisir une valeur par défaut adaptée pour un champ en spécifiant une valeur nulle. Si le *Serveur* n'est pas en mesure de sélectionner une valeur par défaut adaptée, l'entrée correspondante dans la matrice operationResults pour l'*Événement* concerné doit être *Bad_InvalidArgument*.

6.8.4.3 Fonctionnalité de remplacement d'événement

Cette fonction est censée remplacer des champs dans les entrées d'*Événement* existantes (données d'*Événement* correctes contenues dans des données incorrectes en raison d'un capteur défectueux, par exemple).

Le paramétrage performInsertReplace = REPLACE_2 remplace des entrées dans la base de données historique d'*Événement* pour les *EventIds* spécifiés d'un ou de plusieurs *HistoricalEventNodes*. Le paramètre *SelectClause* de l'*EventFilter* doit spécifier la *Propriété EventId*, et l'*eventData* doit contenir l'*EventId* qui sera utilisé pour rechercher l'*Événement* à remplacer. En l'absence de donnée correspondant à l'*EventId* spécifié, aucune opération de remplacement n'est réalisée. L'entrée *operationResults* de l'*eventData* doit plutôt indiquer *Bad_NoEntryExists*. Le paramètre *whereClause* de l'*EventFilter* doit être vide.

Si la *selectClause* spécifie des champs qui ne sont pas valides pour l'*EventType* ou qui ne peuvent pas être sauvegardés ni modifiés par l'historique, l'*entrée operationResults* de l'*Événement* concerné doit indiquer *Good_DataIgnored*. Des informations supplémentaires relatives aux champs ignorés doivent être fournies par l'intermédiaire du *DiagnosticInformation* lié aux *operationResults*. Le symbolicId contient l'indice de chaque champ ignoré séparé par un espace, le localizedText contenant les noms symboliques des champs ignorés.

Si une erreur fatale se produit lors du traitement des champs individuels, l'entrée *operationResults* de l'Événement concerné doit indiquer *Bad_InvalidArgument*, et les champs non valides doivent être indiqués dans le *DiagnosticInformation* lié à l'*entrée operationResults*.

6.8.4.4 Fonctionnalité de mise à jour d'événement

Cette fonction est censée insérer/remplacer sans condition des *Événements* (la synchronisation d'une base de données d'*Événement* de sauvegarde, par exemple).

Le paramétrage performInsertReplace = UPDATE_3 permet d'insérer ou de remplacer des entrées dans la base de données historique d'*Événement* pour le filtre spécifié d'un ou de plusieurs *HistoricalEventNodes*.

En fonction de ses propres critères, le *Serveur* tente de déterminer si l'*Événement* existe déjà. Si c'est le cas, l'*Événement* est supprimé et le nouvel *Événement* inséré (en conservant l'*EventId*). Si l'*EventID* a été fourni, l'*EventID* est utilisé pour déterminer si l'*Événement* existe déjà. Si l'*Événement* n'existe pas, un nouvel *Événement* est inséré, incluant la génération d'un nouvel *EventId*.

Toutes les restrictions, tous les comportements et toutes les erreurs spécifiés pour la fonctionnalité d'insertion (voir 6.8.4.2) s'appliquent également à cette fonction.

Si une entrée d'*Événement* existante a été remplacée avec succès, l'*entrée operationResults* connexe doit être *Good_EntryReplaced*. Si une entrée d'*Événement* a été créée, l'entrée *operationResults* connexe doit être *Good_EntryInserted.* Si le *Serveur* ne peut pas déterminer s'il a remplacé ou inséré une entrée, l'*entrée operationResults* connexe doit être *Good.*

6.8.5 Structure DeleteRawModifiedDetails

6.8.5.1 Détails de la structure DeleteRawModifiedDetails

Le Tableau 32 définit la structure DeleteRawModifiedDetails.

Tableau 32 – DeleteRawModifiedDetails

Nom	Type	Description
DeleteRawModifiedDetails	Structure	Détails de la suppression des valeurs brutes et des mises à jour modifiées.
nodeId	NodeId	ID de nœud de l'*Objet* dont les valeurs historiques sont à supprimer.
isDeleteModified	Booléen	TRUE pour MODIFIED, FALSE pour RAW. La valeur par défaut est FALSE.
startTime	UtcTime	Début de la période à supprimer
endTime	UtcTime	Fin de la période à supprimer

Ces fonctions sont censées être utilisées pour supprimer des données ayant été entrées par erreur dans la base de données historique (suppression de données d'une source avec des horodatages incorrects, par exemple). *startTime* et *endTime* doivent être définies. *startTime* doit être inférieure à *endTime*, les valeurs inférieures mais pas égales à *endTime* étant supprimées. *startTime* = *endTime* est admis, auquel cas la valeur en *startTime* est supprimée.

6.8.5.2 Fonctionnalité DeleteRaw

Le paramétrage isDeleteModified = FALSE permet de supprimer toutes les entrées *Raw* de la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*.

Si aucune donnée n'est trouvée dans l'intervalle de temps pour un *HistoricalDataNode* particulier, le *StatusCode* de cet élément est *Bad_NoData.*

6.8.5.3 Fonctionnalité DeleteModified

Le paramétrage isDeleteModified = TRUE permet de supprimer toutes les entrées *Modified* de la base de données historique pour le domaine temporel spécifié d'un ou de plusieurs *HistoricalDataNodes*.

Si aucune donnée n'est trouvée dans l'intervalle de temps pour un *HistoricalDataNode* particulier, le *StatusCode* de cet élément est *Bad_NoData.*

6.8.6 Structure DeleteAtTimeDetails

6.8.6.1 Détails de la structure DeleteAtTimeDetails

Le Tableau 33 définit la structure de DeleteAtTimeDetails.

Tableau 33 – DeleteAtTimeDetails

Nom	Type	Description
DeleteAtTimeDetails	Structure	Détails des mises à jour d'historiques DeleteRaw.
nodeId	NodeId	ID de nœud de l'*Objet* dont les valeurs historiques sont à supprimer.
reqTimes []	UtcTime	Les entrées définissent les horodatages spécifiques des valeurs à supprimer.

6.8.6.2 Fonctionnalité DeleteAtTime

La structure DeleteAtTime permet de supprimer toutes les entrées de la base de données historique pour les horodatages spécifiés d'un ou de plusieurs *HistoricalDataNodes*.

Ce paramètre est censé être utilisé pour supprimer des données spécifiques de la base de données historique (les données de laboratoire incorrectes et qui ne peuvent pas être correctement reproduites, par exemple).

6.8.7 Structure DeleteEventDetails

6.8.7.1 Détails de la structure DeleteEventDetails

Le Tableau 34 définit la structure de DeleteEventDetails.

Tableau 34 – DeleteEventDetails

Nom	Type	Description
DeleteEventDetails	Structure	Détails de la suppression des valeurs brutes et des mises à jour modifiées.
nodeId	NodeId	ID de nœud de l'*Objet* dont les valeurs historiques sont à supprimer.
eventId[]	ByteString	Matrice des *EventIds* visant à identifier les *Événements* à supprimer.

6.8.7.2 Fonctionnalité de suppression d'événement

La structure DeleteEventDetails permet de supprimer toutes les entrées d'*Événement* de la base de données historique correspondant à l'*EventId* d'un ou de plusieurs *HistoricalEventNodes*.

Si aucun *Événement* correspondant au filtre spécifié n'est trouvé pour un *HistoricalEventNode*, le *StatusCode* de ce *Nœud* est *Bad_NoData.*

Annexe A
(informative)

Conventions du client

A.1 Comment les clients peuvent demander des horodatages

Les spécifications basées sur OPC HDA COM permettaient à un *Client* de demander de manière programmée une période historique en durée absolue (Jan 01, 2006 12:15:45) ou une représentation en chaîne d'heures relatives (NOW -5M). La spécification OPC UA ne permet pas d'utiliser une représentation en chaîne pour transmettre des informations de date/heure à l'aide des *Services* normalisés.

Les applications *Clients* OPC UA qui souhaitent représenter visuellement la date/l'heure au format de chaîne relatif doivent convertir ce format de chaîne en valeurs UTC DateTime avant d'envoyer les requêtes au *Serveur* UA. Il est recommandé à tous les *Clients* OPC UA d'utiliser la syntaxe définie dans l'Article A.1 pour représenter les durées relatives dans leurs interfaces utilisateurs.

Le format de la durée relative est le suivant:

```
keyword+/-offset+/-offset…
```

où "keyword" et "offset" sont spécifiés au Tableau A.1 ci-dessous. Les espaces sont ignorés. La chaîne temporelle doit commencer par un mot-clé. Chaque décalage doit être précédé d'un entier signé qui spécifie le nombre et le sens du décalage. Si l'entier qui précède le décalage n'est pas signé, la valeur du signe précédent est supposée (le signe par défaut de début est positif). Le mot-clé fait référence au début de la période de temps spécifiée. DAY indique l'horodatage au début du jour courant (00:00 heures, minuit). MONTH indique l'horodatage au début du mois courant, etc.

Par exemple, "DAY -1D+7H30M" peut représenter l'heure de début des données demandées pour un rapport quotidien commençant le jour précédent à 7:30 le matin du jour précédent (DAY = premier horodatage du jour courant, -1D est le premier horodatage de la veille, +7H serait 7 heures du matin la veille, +30M serait 7:30 heures du matin la veille (le signe + du dernier terme est étendu depuis le dernier terme).

De la même manière, "MONTH-1D+5H" serait 5 heures du matin le dernier jour du mois précédent, "NOW-1H15M" serait il y a une heure et quinze minutes, et "YEAR+3MO" serait le premier horodatage du 1er avril de cette année.

La résolution des horodatages relatifs repose sur ce que Microsoft a fait avec Excel. Par conséquent, les résultats suivants sont obtenus pour différentes chaînes temporelles douteuses:

10-Jan-2001 + 1 MO = 10-Feb-2001

29-Jan-1999 + 1 MO = 28-Feb-1999

31-Mar-2002 + 2 MO = 30-May-2002

29-Feb-2000 + 1 Y = 28-Feb-2001

Dans la gestion des écarts du calendrier (les mois ou années ne contenant pas toujours le même nombre de jours), lorsque des mois ou des années sont ajoutés ou soustraits:

Mois: si la réponse tombe dans l'écart, elle est sauvegardée jusqu'à la même heure du dernier jour du mois.

Année: si la réponse tombe dans l'écart (le 29 février), elle est sauvegardée jusqu'à la même heure du jour le 28 février.

Noter que cela n'est pas vrai en cas d'ajout ou de retrait de semaines ou de jours, mais uniquement pour l'ajout ou le retrait de mois ou d'années pouvant contenir un nombre de jours différents.

Noter que tous les mots-clés et décalages sont spécifiés en majuscules (voir Tableaux A.1 et A.2).

Tableau A.1 – Définitions de mot-clé temporel

Mot-clé	Description
NOW	Heure UTC en cours calculée sur le *Serveur*.
SECOND	Début de la seconde en cours.
MINUTE	Début de la minute en cours.
HOUR	Début de l'heure en cours.
DAY	Début du jour en cours.
WEEK	Début de la semaine en cours.
MONTH	Début du mois en cours.
YEAR	Début de l'année en cours.

Tableau A.2 – Définitions de décalage temporel

Décalage	Description
S	Décalage temporel en secondes.
M	Décalage temporel en minutes.
H	Décalage temporel en heures.
D	Décalage temporel en jours.
W	Décalage temporel en semaines.
MO	Décalage temporel en mois.
Y	Décalage temporel en années.

A.2 Détermination du premier point de données historiques

Dans certains cas, les *Serveurs* nécessitent de renvoyer le premier point de données disponible pour un *Nœud* historique. L'Article A.2 recommande la manière dont il convient qu'un *Client* demande ces informations, de sorte que le *Serveur* puisse optimiser cet appel, le cas échéant. Même si plusieurs appels peuvent renvoyer la première valeur de données, la pratique recommandée consiste à utiliser la *Propriété StartOfArchive*. Si cette *Propriété* n'est pas disponible, utiliser les paramètres ReadRawModifiedDetails suivants:

returnBounds=false

numValuesPerNode=1

startTime=DateTime.MinValue+1 seconde

endTime= DateTime.MinValue

Bibliography

IEC TR 62541-2, *OPC Unified Architecture – Part 2: Security Model* (disponible en anglais seulement)

IEC 62541-6, *Architecture unifiée OPC – Partie 6: Correspondances*

IEC 62541-7, *Architecture unifiée OPC – Partie 7: Profils*

IEC 62541-9, *Architecture unifiée OPC – Partie 9: Alarmes et conditions*

_____

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch