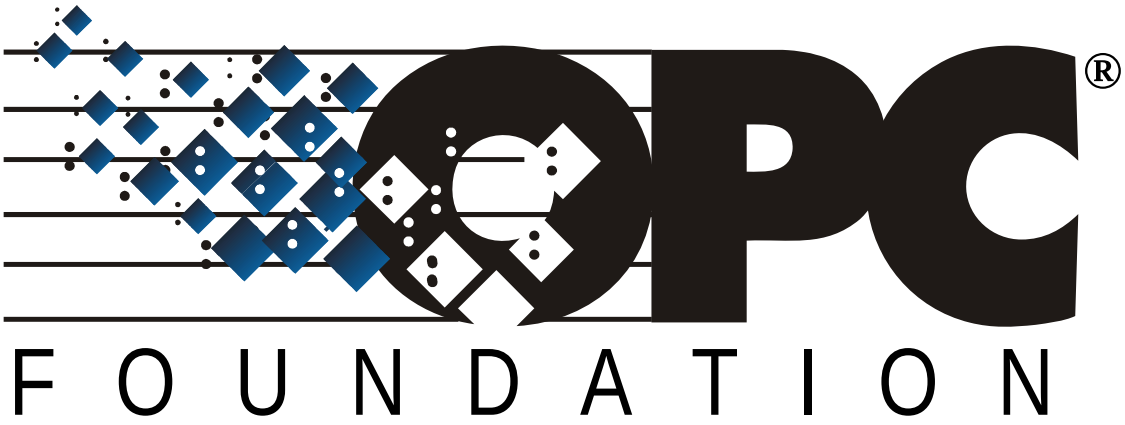


OPC Unified Architecture

Specification

Part 11: Historical Access

Version 1.00

January 5, 2007

Specification Type	Industry Standard Specification		
Title:	OPC Unified Architecture Historical Access	Date:	January 5, 2007
Version:	Release 1.00	Software Source:	MS-Word OPC UA Part 11 - Historical Access 1.00 Specification.doc
Author:	OPC Foundation	Status:	Release

CONTENTS

Page

FIGURES

TABLES

OPC FOUNDATION

UNIFIED ARCHITECTURE –

FOREWORD

This specification is for developers of OPC UA clients and servers. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of servers and clients by multiple vendors that shall inter-operate seamlessly together.

Copyright © 2007, OPC Foundation, Inc.

AGREEMENT OF USE

COPYRIGHT RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

OPC Foundation members and non-members are prohibited from copying and redistributing this specification. All copies must be obtained on an individual basis, directly from the OPC Foundation Web site http://www.opcfoundation.org.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OPC specifications may require use of an invention covered by patent rights. OPC shall not be responsible for identifying patents for which a license may be required by any OPC specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OPC specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

WARRANTY AND LIABILITY DISCLAIMERS

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OPC FOUDATION MAKES NO WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OPC FOUNDATION BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you.

RESTRICTED RIGHTS LEGEND

This Specification is provided with Restricted Rights. Use, duplication or disclosure by the U.S. government is subject to restrictions as set forth in (a) this Agreement pursuant to DFARs 227.7202-3(a); (b) subparagraph (c)(1)(i) of the Rights in Technical Data and Computer Software clause at DFARs 252.227-7013; or (c) the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 subdivision (c)(1) and (2), as applicable. Contractor / manufacturer are the OPC Foundation,. 16101 N. 82nd Street, Suite 3B, Scottsdale, AZ, 85260-1830

COMPLIANCE

The OPC Foundation shall at all times be the sole entity that may authorize developers, suppliers and sellers of hardware and software to use certification marks, trademarks or other special designations to indicate compliance

with these materials. Products developed using this specification may claim compliance or conformance with this specification if and only if the software satisfactorily meets the certification requirements set by the OPC Foundation. Products that do not meet these requirements may claim only that the product was based on this specification and must not claim compliance or conformance with this specification.

TRADEMARKS

Most computer and software brand names have trademarks or registered trademarks. The individual trademarks have not been listed here.

GENERAL PROVISIONS

Should any provision of this Agreement be held to be void, invalid, unenforceable or illegal by a court, the validity and enforceability of the other provisions shall not be affected thereby.

This Agreement shall be governed by and construed under the laws of the State of Minnesota, excluding its choice or law rules.

This Agreement embodies the entire understanding between the parties with respect to, and supersedes any prior understanding or agreement (oral or written) relating to, this specification.

ISSUE REPORTING

The OPC Foundation strives to maintain the highest quality standards for its published specifications, hence they undergo constant review and refinement. Readers are encouraged to report any issues and view any existing errata here:
http://www.opcfoundation.org/errata

1 Scope

This specification is part of the overall OPC Unified Architecture specification 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 [UA Part 3]. The predefined information model is defined in [UA Part 5]. The services to detect and access historical data and events, and description of the *ExtensibleParameter* types are specified in [UA Part 4].

2 Reference documents

[UA Part 1] OPC UA Specification: Part 1 – Concepts, Version 1.0 or later
http://www.opcfoundation.org/UA/Part1/

[UA Part 3] OPC UA Specification: Part 3 – Address Space Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part3/

[UA Part 4] OPC UA Specification: Part 4 – Services, Version 1.0 or later
http://www.opcfoundation.org/UA/Part4/

[UA Part 5] OPC UA Specification: Part 5 – Information Model, Version 1.0 or later
http://www.opcfoundation.org/UA/Part5/

[UA Part 7] OPC UA Specification: Part 7 – Profiles, Version 1.0 or later
http://www.opcfoundation.org/UA/Part7/

[UA Part 8] OPC UA Specification: Part 8 – Data Access, Version 1.0 or later
http://www.opcfoundation.org/UA/Part8/

[UA Part 9] OPC UA Specification: Part 9 – Alarm & Conditions, Version 1.0 or later
http://www.opcfoundation.org/UA/Part9/

3 Terms, definitions, and abbreviations

3.1 OPC UA Part 1 terms

The following terms defined in [UA Part 1] apply.

1) AddressSpace
2) Attribute
3) BrowseName
4) Event
5) Node
6) NodeId
7) Notification
8) Object
9) ObjectType

3.2 OPC UA Part 3 terms

The following terms defined in [UA Part 3] apply.

1) DataVariable
2) EventType
3) Property
4) Variable

3.3 OPC UA Part 4 terms

The following terms defined in [UA Part 4] apply.

1) ExtensibleParameter
2) StatusCode
3) ServerTimestamp
4) SourceTimestamp

3.4 OPC UA Historical Access terms

3.4.1 Aggregate

Provides summarized data values of process data.

An *Aggregate* is a way to produce a set of values derived from the raw data in the historian. Clients may specify an *Aggregate* when using the *ReadHistory* service. Complete details of the various standard *Aggregates* and their behaviour are outlined in Clause 5.6. Common *Aggregates* include averages over a given time range, minimum over a time range and maximum over a time range.

3.4.2 Annotation

An *Annotation* is a user entered comment that is associated with an item at a given instance in time. There does not have to be a value stored at that time.

3.4.3 BoundingValues

BoundingValues are the values that are associated with the starting and ending time of an interval specified when reading from the historian. *BoundingValues* are 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 Clause 4.4 for details on using *BoundingValues*.

3.4.4 HistoricalNode

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* where *HistoricalNode* is an *Object*, *Variable*, *Property* or *View*. The term *HistoricalNode* refers to both *HistoricalDataNodes* and *HistoricalEventNodes*, and is used when referencing aspects of the specification that apply to accessing historical data and events.

3.4.5 HistoricalDataNode

A *HistoricalDataNode* represents any *Variable* or *Property* in the *AddressSpace* for which a client may read and/or update historical data. The terms "*HistoricalDataNode's* history" or "history of a *HistoricalDataNode*" will refer to the time series data stored for this *HistoricalNode* where *HistoricalNode* is an *Object*, *Variable*, *Property* or *View*. Some 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 specification that apply to accessing historical data only.

3.4.6 HistoricalEventNode

A *HistoricalEventNode* represents any *Object* or *View* in the *AddressSpace* for which a client may read and/or update historical events. The terms "*HistoricalEventNode's* history" or "history of a *HistoricalEventNode*" will refer to the time series events stored in some historical system. Some 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 specification that apply to accessing historical events only.

3.4.7 Interpolated data

Interpolated data is data that is calculated from the data in the archive. An interpolated value is calculated from the stored data points on either side of the requested timestamp.

3.4.8 Modified values

A modified value is a *HistoricalDataNode's* value that has been changed (or deleted) after it was stored in the historian. 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. 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.

3.4.9 Raw data

Raw data is data that is stored within the historian for a *HistoricalDataNode* . The data may be all data collected for the *DataItem* or it may be some subset of the data depending on the historian and the storage rules invoked when the item values were saved.

3.4.10 StartTime / EndTime

The *StartTime* and *EndTime* specify the bounds of a history request and define the time domain of the request. 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 archive exactly once. See the examples in Clause 5.6.2.2

3.4.11 TimeDomain

The interval of time covered by a particular request, or by a particular response. 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 section 4.4. *BoundingValues* effect the time domain as described in section 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 [UA Part 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 clients

3.5 Abbreviations and symbols

DA	Data Access
DCS	Distributed Control System
HD	Historical Data
PLC	Programmable Logic Controller
UA	Unified Architecture

4 Concepts

4.1 General

The OPC UA Historical Access specification defines the representation 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 OPC UA *AddressSpace* and the definition of aggregates used in processed data retrieval.

4.2 Data representation

An OPC UA Server supporting Historical Access provides one or more OPC UA 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 archive, database or a short term buffer within memory. An OPC UA Server supporting Historical Access may or may not provide historical data and events for some or all available *Variables*, *Objects* or *Views* within the server *AddressSpace*. As with the other information models, the *AddressSpace* of an OPC UA Server supporting Historical Access is accessed via the View or *Query* service sets.

An OPC UA Server supporting Historical Access provides a way to access or communicate to a set of historical data and/or historical event sources. The types of sources available are a function of the server implementation.

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.

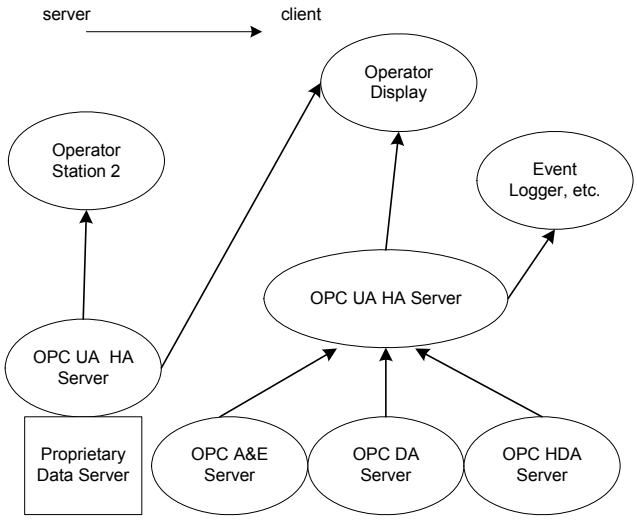


Figure 1 - Possible OPC UA Server supporting Historical Access

The server may be implemented as a stand alone OPC UA Server that collects data from another OPC UA Server, a legacy OPC HDA Server, a legacy OPC DA Server, a legacy OPC A&E Server or another data source. The clients that reference 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 or events, and clients may request which timestamp will be used as the reference. See [UA Part 4] for details on the *TimestampsToReturn* enumeration. An OPC UA Server supporting Historical Access will treat the various timestamp settings as described below.

For *HistoricalDataNodes*:

SOURCE	Return the *SourceTimestamp*. *SourceTimestamp* is used to determine which historical data values are returned.
SERVER	Return the *ServerTimestamp*. *ServerTimestamp* is used to determine which historical data values are returned.
BOTH	Return both the *SourceTimestamp* and *ServerTimestamp*. *SourceTimestamp* is used to determine which historical data values are returned.
NEITHER	This is not a valid setting for any HistoryRead accessing *HistoricalDataNodes.*

For *HistoricalEventNodes*:

SOURCE	Return the *SourceTimestamp*. *SourceTimestamp* is used to determine which historical events are returned.
SERVER	This is not valid setting for any HistoryRead accessing *HistoricalEventNodes.*
BOTH	This is not valid setting for any HistoryRead accessing *HistoricalEventNodes.*
NEITHER	This is not valid setting for any HistoryRead accessing *HistoricalEventNodes.*

Any reference to Timestamps through out this specification will represent either *ServerTimestamp* or *SourceTimestamp* as dictated by the type requested in the ReadHistory service. Some servers may not support historizing both *SourceTimestamp* and *ServerTimestamp*, but it is expect that all servers will support historizing *SourceTimestamp* (see [UA Part 7] for details on Server Profiles).

4.4 Bounding values and time domain

When accessing *HistoricalDataNodes via* the *ReadHistory* Service, requests can set a flag, returnBounds, indicating that a *BoundingValue* are requested. For a complete description of the extensible Parameter *HistoryReadDetails* that include all of these parameters see section 5.3. The concept of bounding values and how they affect the time domain that is requested as part of the *ReadHistory* request is further explained in this section. This section also provides examples of *TimeDomains* to further illustrate the expected behaviour.

When making a request for historical data using the *ReadHistory* Service, required parameters include a *startTime* and *endTime*. These two parameters define the *TimeDomain* of the ReadHistory request. This *TimeDomain* includes all values between the *StartTime* and *EndTime*, and any value that falls exactly on the *StartTime*, but not any value that falls exactly on the *EndTime*. For example, assuming bounding values are not requested, if data is requested from 1:00 to 1:05, and then from 1:05 to 1:10, a value that exists at exactly 1:05 would be included in the second request, but not in the first.

Given that a historian has values stored at 5:00, 5:02, 5:03, 5:05 and 5:06, the data returned from a RAW data call is given by Table 1. In the table, FIRST stands for a tuple with a value of *DateTime.Min*, a timestamp of the specified StartTime, and a *StatusCode* of *Bad_NoBound*. LAST stands for a tuple with a value of *DateTime.Max*, a timestamp of the specified EndTime, and a *StatusCode* of *Bad_NoBound*

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	DateTime.Max	3	Yes	5:00, 5:02, 5:03
5:00	DateTime.Max	3	No	5:00, 5:02, 5:03
5:00	DateTime.Max	6	Yes	5:00, 5:02, 5:03, 5:05, 5:06, LAST
5:00	DateTime.Max	6	No	5:00, 5:02, 5:03, 5:05, 5:06
DateTime.Min	5:06	3	Yes	5:06,5:05,5:03
DateTime.Min	5:06	3	No	5:06,5:05,5:03
DateTime.Min	5:06	6	Yes	5:06,5:05,5:03,5:02,5:00,FIRST
DateTime.Min	5:06	6	No	5:06, 5:05, 5:03, 5:02, 5:00
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
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
5:01	5:01	1	Yes	5:00
5:01	5:01	1	No	NODATA

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 *Properties* supported by one or more specified *Nodes*. See [UA Part 4] 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 an active point, the object should be maintained in the address space, with the appropriate *Access Level* attribute and Historizing attribute settings (see [UA Part 3] for details on access levels).

4.6 Historical Audit Events

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*.

Servers must generate events of the *AuditUpdateEventType* or a sub-type of this type for all invocations of the *HistoryUpdate* service on any *HistoricalNode*. See [UA Part 3] and [UA Part 5] for details on the *AuditUpdateEventType* Model. In the case where the *HistoryUpdate* service is invoked to insert HistoricalEvents, the AuditUpdateEvent must include the EventId of the inserted event and a description that indicates that the event was inserted. All other updates must follow the guidelines provided in the *AuditUpdateEventType* Model.

4.7 Model

4.7.1 HistoricalDataNodes

4.7.1.1 General

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

4.7.1.2 HasHistoricalConfiguration

The *HasHistoricalConfiguration ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantic of this *ReferenceType* is to bind a *DataVariable* or a *Property* to its *HistoricalConfigurationType Object.* All *DataVariables* and *Properties* that expose historical data must have exactly one *HasHistoricalConfiguration* reference.

The *SourceNode* of this *ReferenceType* must be a *DataVariable or Property*. The *TargetNode* must be an *Object* of the *ObjectType HistoricalConfigurationType*.

Multiple *DataVariables* or *Properties* may reference the same *HistoricalConfigurationType Object*.

4.7.1.3 OptionalNew

ModellingRules are an extendable concept in OPC UA; [UA Part 3] defines the rules "None", "Shared" and "New". Some Historical Access properties, however, are optional and this part therefore also uses *OptionalNew ModellingRule*. This *ModellingRule* is defined in [UA Part 8]

4.7.1.4 HistoricalConfigurationType

The Historical Access Data model extends the standard type model by defining an additional *ObjectType,* the *HistoricalConfigurationType* This *HistoricalConfigurationType* defines the general characteristics of a node that defines the historical configuration of any variable or property that is defined to contain history. It is formally defined in Table 2.

Table 2 – HistoricalConfigurationType Definition

Attribute	Value				
BrowseName	HistoricalConfigurationType				
IsAbstract	False				
References	**NodeClass**	**BrowseName**	**DataType**	**TypeDefinition**	**ModellingRule**
Subtype of the *BaseObjectType* defined in [UA Part 5]					
HasProperty	Variable	Stepped	Boolean	PropertyType	New
HasProperty	Variable	Definition	String	PropertyType	Optional New
HasProperty	Variable	MaxTimeInterval	Duration	PropertyType	Optional New
HasProperty	Variable	MinTimeInterval	Duration	PropertyType	Optional New
HasProperty	Variable	ExceptionDeviation	Double	PropertyType	Optional New
HasProperty	Variable	ExceptionDeviationFormat	Enum	PropertyType	Optional New

Stepped specifies whether the historical data was collected in such a manner that it should be displayed as interpolated (sloped Lines between point) or as Stepped (vertically-connected horizontal lines between points) when raw data is examined. This property also effect how some aggregates are calculated. A value of True indicates stepped mode. A value of False indicates interpolated mode. The default value is False.

Definition 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”

MaxTimeInterval specifies the maximum interval between data points in the history repository regardless of their value change (see [UA Part 4] for definition of *Duration*).

MinTimeInterval specifies the minimum interval between data points in the history repository regardless of their value change (see [UA Part 4] for definition of *Duration*).

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

ExceptionDeviationFormat specifies how the ExceptionDeviation is determined. Its values are defined in Table 3.

Table 3 – ExceptionDeviationFormat Values

Value	Description
ABSOLUTE_VALUE_0	ExceptionDeviation is an absolute Value.
PERCENT_OF_RANGE_1	ExceptionDeviation is a percent of InstrumentRange (See [UA Part 8])
PERCENT_OF_VALUE_2	ExceptionDeviation is a percent of Value.

4.7.2 HistoricalDataNodes Address Space Model

HistoricalDataNodes are always 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". But it can be an *Object* of any other type.

Figure 2 illustrates the basic AddressSpace model of a DataVariable that includes History.

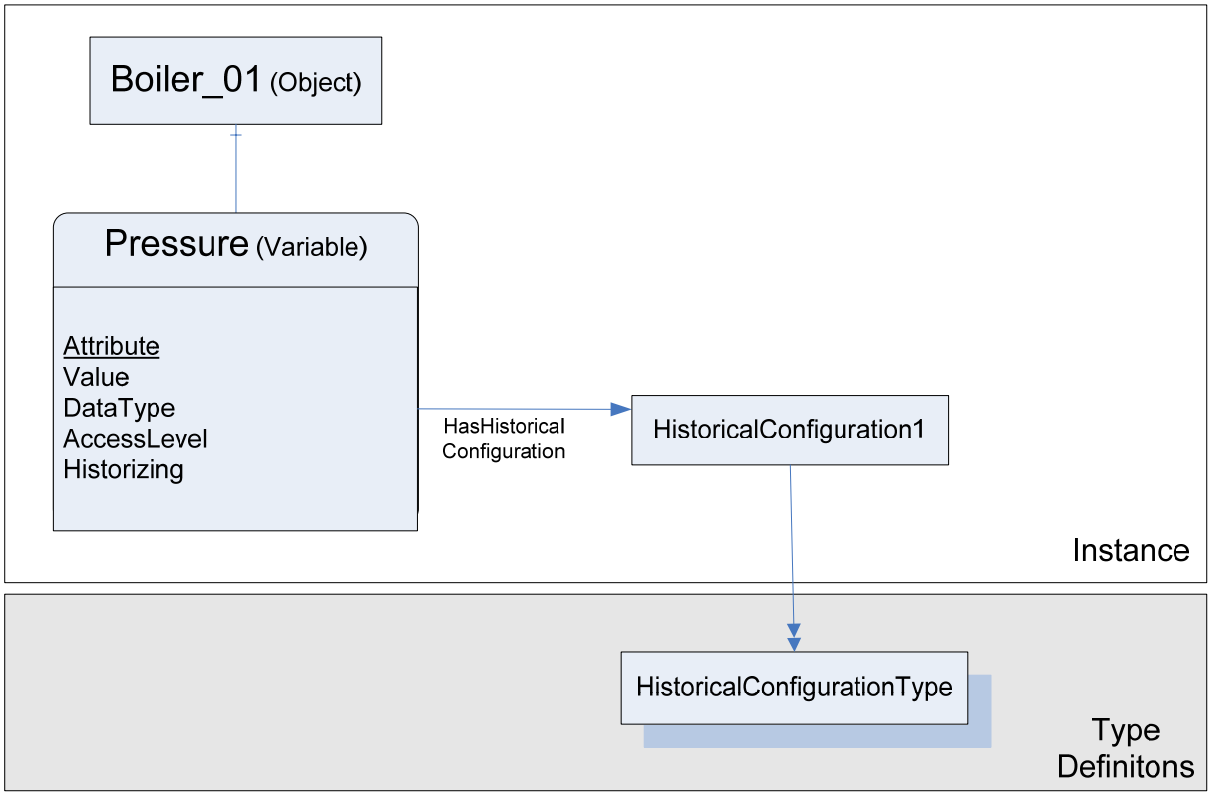


Figure 2 – Representation of a *Variable* with History in the AddressSpace

Each *Variable* with history must have the *Historizing* attribute (see [UA Part 3]*)* defined and include a *HasHistoricalConfiguration* reference. The *HistoricalConfigurationType* Instance must define the stepped property, but may also define any of the optional properties.

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 [UA Part 3] for details on use of this *Attribute).*

Figure 2 only shows a subset of *Attributes* and *Properties*. Other *Attributes* as that are defined for *Variables* in [UA Part 3], and in the following sections – may also be available.

4.7.3 HistoricalDataNodes Attributes

This section lists the *Attributes* of *Variables* that have particular importance for historical data. They are specified in detail in [UA Part 3]. The following *Attributes* are particularly important for *HistoricalDataNodes*.

- Value
- DataType
- AccessLevel
- Historizing

Value is the value of the *Variable.* Its data type is defined by the *DataType Attribute*. This is the *Attribute* for which historical data is collected. The *AccessLevel* attribute defines the server's basic ability to access history data for this *Variable*.

When a client requests the *Value* attribute, the server in addition always returns a *StatusCode* (the quality and the server's ability to access/provide the value) and a *ServerTimestamp* and/or a *SourceTimestamp*. See [UA Part 4] for details on *StatusCode* and the meaning of the two timestamps. Specific *StatusCodes* for *HistoricalDataNodes* are defined in Clause 5.2.

4.7.4 HistoricalEventNodes

4.7.4.1 General

The Historical Event model defines additional *ReferenceTypes*, *ModellingRules* and *ObjectTypes*. These descriptions also include required use cases for *HistoricalEventNodes.*

4.7.4.2 HasHistoricalEventConfiguration

The *HasHistoricalEventConfiguration ReferenceType* is a concrete *ReferenceType* and can be used directly. It is a subtype of *NonHierarchicalReferences*.

The semantics of this *ReferenceType* is to bind an *Object* which exposes an *EventNotifier* that exposes historical events (i.e. has the *EventNotifier Attribute* for HistoryRead or HistoryWrite set to one) to a *HistoricalEventConfigurationType Object*. All objects which expose *EventNotifiers* that expose historical events must have exactly one *HasHistoricalEventConfiguration* reference.

The *SourceNode* of this *ReferenceType* must be an *Object* which exposes a *EventNotifier* that exposes historical events. The *TargetNode* must be an *Object* of the *ObjectType HistoricaEventIConfigurationType*.

Multiple *EventNotifiers* may reference the same *HistoricalEventConfigurationType* Object.

4.7.4.3 OptionalNew

ModellingRules are an extendable concept in OPC UA; [UA Part 3] defines the rules "None", "Shared" and "New". Some Historical Access properties, however, are optional and this part therefore also uses *OptionalNew ModellingRule*. This *ModellingRule* is defined in [UA Part 8]

4.7.4.4 HistoricalEventConfigurationType

The Historical Access Event model extends the standard type model by defining an additional *ObjectType,* the *HistoricalEventConfigurationType* This *HistoricalEventConfigurationType* defines the general characteristics of a node that defines the historical configuration of any *Object* that exposes an *EventNotifier* that exposes historical events. It is formally defined in Table 4

Table 4 – HistoricalEventConfigurationType Definition

Attribute	**Value**		
BrowseName	HistoricalEventConfigurationType		
IsAbstract	False		
References	**NodeClass**	**BrowseName**	**ModellingRule**
Subtype of the *BaseObjectType* defined in [UA Part 5]			
GeneratesEvent	ReferenceType	HistoricalEvents01	OptionalNew

HistoricalEvents: The semantic of this ReferenceType is to relate EventTypes that are being historized to the object that they are available from. This *ReferenceType* and any subtypes

are intended to be used for discovery of types of historical *Events* in a server. They are not required to be present for a server to historize *Events*. This *ReferenceType* is as described in [UA Part 3]. This application of this *ReferenceType* further restricts the use as follows:

The *SourceNode* of this *ReferenceType* must be a *Node* that is of type *HistoricalEventConfigurationType*

The *TargetNode* of this *ReferenceType* must be the *EventType* that is available as historical events.

The *Object* of *HistoricalEventConfigurationType* can expose more then one of these references. The resulting list of E*ventType Nodes* (and there sub types) is the summary list the types of *Events* that are available as historical events. A server that does not historize all attributes associated with a given E*ventType* should define a new E*ventType* that describes the attributes that are being historized and add a *Reference* to it from it's *HistoricalEventConfigurationType Object*. The *BrowseName* of the reference can be any name that is unique for the *Object* of *HistoricalEventConfigurationType* and follows the naming requirements of *BrowseNames*. A user should review all *GenerateEvent* references in the *Object* of *HistoricalEventConfigurationType* that is associated with the *Object* that is exposing historical events

4.7.5 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*. Additional characteristics of *HistoricalEventNodes* are defined using *Properties* (i.e. *Variables* that are referenced using *HasProperty References*). For a detailed description of *Variable* and *Properties* see [UA Part 3]. This specification defines *Properties* that have been found useful for a large range of historical event clients.

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 [UA Part 3] for details on use of this *Attribute.*

Figure 3 illustrates the basic *AddressSpace* model of an *Event* that includes History.

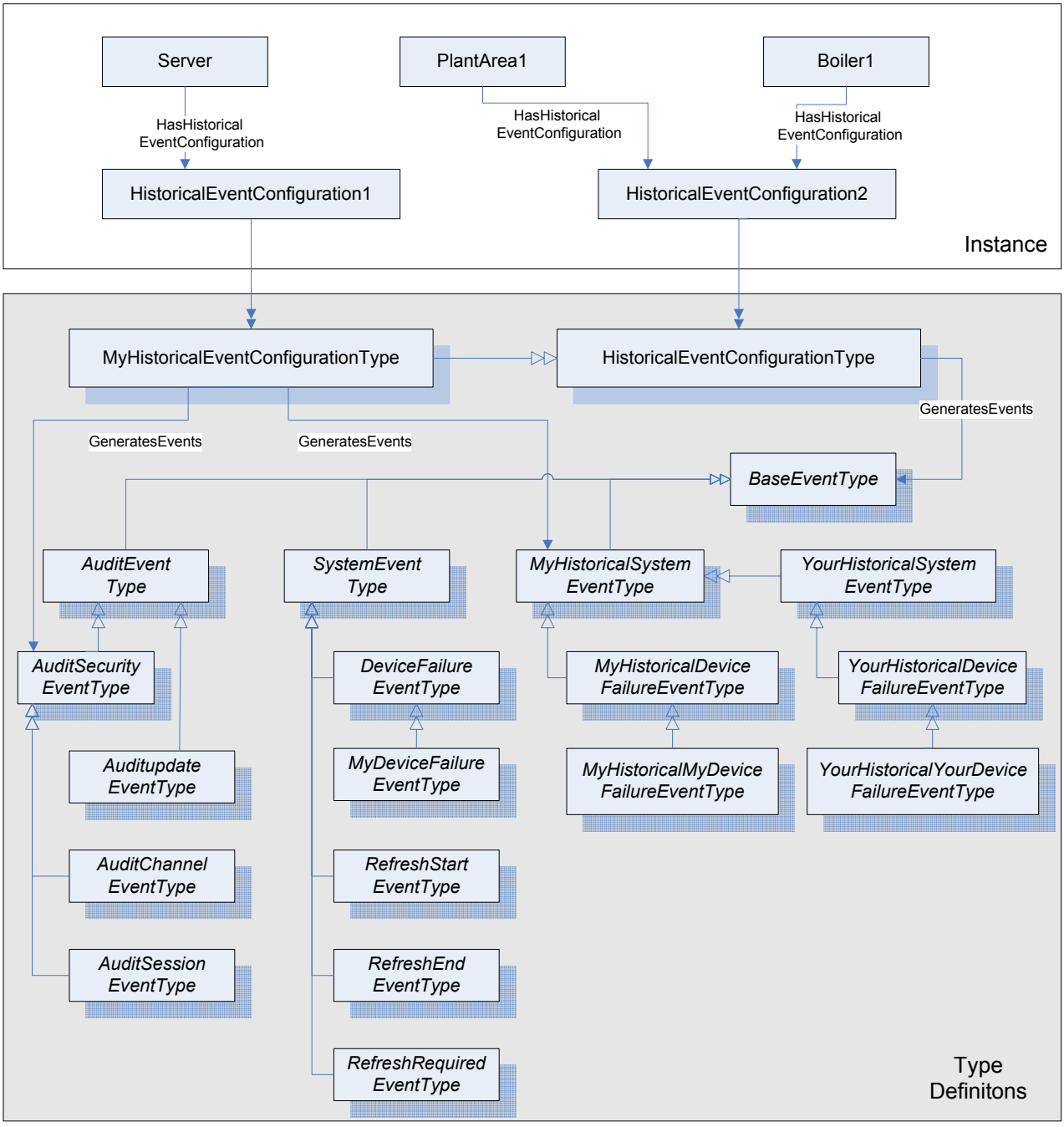


Figure 3 – Representation of an *Event* with History in the AddressSpace

4.7.6 HistoricalEventNodes Attributes

This section lists the *Attributes* of *Objects* or *Views* that have particular importance for historical events. They are specified in detail in [UA Part 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.

4.8 History Objects

4.8.1 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.

4.8.2 HistoryServerCapabilitiesType

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

The content of this *BaseObjectType* is already defined by its type definition in [UA Part 5]. The *Object* extensions are formally defined in Table 5.

Table 5 – HistoryServerCapabilitiesType Definition

Attribute	Value				
BrowseName	HistoryServerCapabilitiesType				
IsAbstract	False				
ArraySize	-1				
References	**Node Class**	**Browse Name**	**Data Type**	**Type Definition**	**Instantiation Rule**
HasProperty	Variable	AccessHistoryDataCapability	Boolean	PropertyType	New
HasProperty	Variable	AccessEventsCapability	Boolean	PropertyType	New
HasProperty	Variable	MaxReturnValues	UInt32	PropertyType	New
HasProperty	Variable	TreatUncertainAsBad	Boolean	PropertyType	New
HasProperty	Variable	PercentDataBad	UInt8	PropertyType	New
HasProperty	Variable	PercentDataGood	UInt8	PropertyType	New
HasProperty	Variable	SteppedInterpolationMode	Boolean	PropertyType	New
HasProperty	Variable	InsertDataCapability	Boolean	PropertyType	New
HasProperty	Variable	ReplaceDataCapability	Boolean	PropertyType	New
HasProperty	Variable	UpdateCapability	Boolean	PropertyType	New
HasProperty	Variable	DeleteRawCapability	Boolean	PropertyType	New
HasProperty	Variable	DeleteAtTimeCapability	Boolean	PropertyType	New
HasProperty	Object	HistoryAggregates	--	HistoryAggregateContainerType	New

All UA server that support Historical data access must include the HistoryServerCapabilities as part of its ServerCapabilities. If any of these *Properties* do not contain a valid value, the client application should use the default values.

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

The *MaxReturnValues Variable* defines 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.

The *TreatUncertainAsBad Variable* indicates how the server treats data returned with a *StatusCode* severity *Uncertain* with respect to aggregate calculations. A value of True indicates the server considers the severity equivalent to *Bad*, a value of False indicates the server considers the severity equivalent to *Good*. The default value is True.

The *PercentDataBad Variable* indicates the Maximum percentage of bad data in a given interval above which would cause the *StatusCode* for the given interval for processed data request to be set to *Bad*. (*Uncertain* is treated as defined above). For values equal to or below this percentage the ***StatusCode*** would be Uncertain or Good. For details on which aggregates use the *PercentDataBad Variable*, see the definition of each aggregate. The default value is 0.

The *PercentDataGood Variable* indicates the minimum percentage of *Good* data in a given interval which would cause the *StatusCode* for the given interval for the processed data requests to be set to *Good*. For values below this percentage the ***StatusCode*** would be

Uncertain or Bad. For details on which aggregates use the *PercentDataGood Variable*, see the definition of each aggregate. The default value is 100.

The *SteppedInterpolationMode Variable* indicates how the server interpolates data when no boundary value exists (i.e. interpolating into the future from the last known value). A value of False indicates that the server will use a stepped format, and hold the last known value constant. A value of True indicates the server will project the value using straight line interpolation. The default value is False.

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

The *UpdateCapability Variable* indicates support for the Update capability. A value of True indicates the server supports the capability to insert new 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 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 value at a specified time. The default value is False.

The *HistoryAggregates Object* defines the aggregate capabilities supported by the UA server. This Object has 'HasComponent' references to zero or more *HistoryAggregate Objects* which define a specific aggregate supported by the server. The *HistoryAggregate Objects* are instances of the *HistoryAggregateType ObjectType* defined in Clause 4.8.3.

4.8.3 HistoryAggregateContainerType

This *ObjectType* defines a container for the standard OPC UA Server supporting Historical Access and vendor aggregates supported by the UA server. This *ObjectType* is formally defined in Table 6. . All servers that expose aggregates must define this *ObjectType* and define the aggregates that it exposes.

Table 6 – HistoryAggregatesType Definition

Attribute	**Value**				
BrowseName	HistoryAggregateContainerType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**Type Definition**	**Mod. Rule**
Subtype of the *BaseObjectType* defined in [UA Part 5]					

4.8.4 HistoryAggregateType

This *ObjectType* defines an aggregate supported by a UA server. This object is formally defined in Table 7.

Table 7 – HistoryAggregateType Definition

Attribute	**Value**				
BrowseName	HistoryAggregateType				
IsAbstract	False				
References	**Node Class**	**BrowseName**	**DataType**	**Type Definition**	**Mod. Rule**
Subtype of the *BaseObjectType* defined in [UA Part 5]					

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

Table 8 outlines the *BrowseName* and *Description* for the standard aggregates.

Table 8 – Standard HistoryAggregateType BrowseNames

BrowseName	Description
	Interpolation Aggregate
Interpolative	Do not retrieve an aggregate. This is used for retrieving interpolated Values.
	Data Averaging and Summation Aggregates
Average	Retrieve the average data over the resample interval.
TimeAverage	Retrieve the time weighted average data over the resample interval.
Total	Retrieve the sum of the data over the resample interval.
TotalizeAverage	Retrieve the totalized Value (time integral) of the data over the resample interval.
	Data Variation Aggregates
Minimum	Retrieve the minimum Value in the resample interval.
Maximum	Retrieve the maximum Value in the resample interval.
MinimumActualTime	Retrieve the minimum value in the resample interval and the Timestamp of the minimum value.
MaximumActualTime	Retrieve the maximum value in the resample interval and the Timestamp of the maximum value.
Range	Retrieve the difference between the minimum and maximum Value over the sample interval.
	Counting Aggregates
AnnotationCount	Retrieve the number of annotations in the interval.
Count	Retrieve the number of raw Values over the resample interval.
DurationInState0	Retrieve the time (in seconds) a Boolean was in a 0 state
DurationInState1	Retrieve the time (in seconds) a Boolean was in a 1 state
NumberOfTransitions	Retrieve the number of state changes a Boolean value experianced in the interval
	Time Aggregates
Start	Retrieve the Value at the beginning of the resample interval. The time stamp is the time stamp of the beginning of the interval.
End	Retrieve the Value at the end of the resample interval. The time stamp is the time stamp of the end of the interval.
Delta	Retrieve the difference between the first and last Value in the resample interval.
	Data Quality Aggragates
DurationGood	Retrieve the duration (in seconds) of time in the interval during which the data is good.
DurationBad	Retrieve the duration (in seconds) of time in the interval during which the data is bad.
PercentGood	Retrieve the percent of data (0 to 100) in the interval which has good StatusCode.
PercentBad	Retrieve the percent of data (0 to 100) in the interval which has bad StatusCode.
WorstQuality	Retrieve the worst StatusCode of data in the interval.

4.9 History DataType definitions

4.9.1 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 underlying system.
annotationTime	UtcTime	The time the annotation was added. This will probably be different than the SourceTimestamp

Its representation in the *AddressSpace* is defined in Table 10.

Table 10 – Annotation Definition

Attributes	Value
Browse Name	Annotation

5 Historical Access specific usage of Services

5.1 General

[UA Part 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*.

5.2 Historical Nodes StatusCodes

5.2.1 Overview

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

The general structure of the *StatusCode* is specified in [UA Part 4]. It includes a set of common operational result codes which also apply to historical data and/or events.

5.2.2 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 it's 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, was the result interpolated, were all data inputs to a calculation of good quality, etc.

In the following, Table 11 contains codes with *Bad* severity indicating a failure; Table 12 contains codes with *Uncertain* severity indicating that the value has been retrieved under sub-normal conditions. Table 13 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 [UA Part 4] and [UA Part 8].

Table 11 – Bad operation level result codes

Symbolic Id	Description
Bad_NoData	No data exists for the requested time range or event filter
Bad_NoBound	No data found to provide upper or lower bound value.
Bad_DataLost	Data is missing due to collection started / stopped / lost.
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 timestamp format the server does not support (i.e requested ServerTimestamp when server only supports SourceTimestamp)

Table 12 – Uncertain operation level result codes

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

Table 13 – Good operation level result codes

Symbolic Id	Description
Good_NoData	No data exists for the requested time range or event filter.
Good_MoreData	There is more data to be returned than could be returned in a single request.
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

5.2.3 Historian Information Bits

These bits are set only when reading historical data of *HistoricalDataNodes*. They indicate where the data value came from and provide information that affects how the client uses the data value. Table 14 lists the bit settings which indicate the data location (i.e. is the value stored in the underlying data repository, or is the value the result of data aggregation). These bits are mutually exclusive.

Table 14 – Data Location

StatusCode	Description
Raw	A raw data value.
Calculated	A data value which was calculated.
Interpolated	A data value which was interpolated.

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

Table 15 lists the bit settings which indicate additional important information about the data values returned.

Table 15 – Additional Information

StatusCode	Description
Partial	A data value which was calculated with an incomplete interval.
Extra Data	A raw data value that hides other data at the same timestamp.
Multiple Values	Multiple values match the aggregate criteria (i.e. multiple minimum values or multiple worst quality at different timestamps within the same interval)

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

5.2.4 Semantics changed

The *StatusCode* in addition contains an informational bit called *Semantics Changed*. (See [UA Part 4])

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

5.3 HistoryReadDetails parameters

5.3.1 Overview

The *HistoryRead* service defined in [UA Part 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 [UA Part 4] for the definition of *Extensible Parameter*. Table 16 lists the valid values for the *parameterTypeId* parameter which specifies which function the *HistoryRead* service will perform, and what structure will be contained in the *parameterData* field .

Table 16 – HistoryReadDetails parameterTypeId Values

Name	Value	Description	parameterData Structure
EVENTS	1	This parameter selects a set of events from the history database by specifying a filter and a time domain for one or more *Objects* or *Views*.	ReadEventDetails (See Clause 5.3.2)

		This parameter is only valid for *Objects* that have the *EventNotifier* attribute set to TRUE (See [UA Part 3]).	
RAW	2	This parameter selects a set of values from the history database by specifying a time domain for one or more *Variables*	ReadRawModifiedDetails (See Clause 5.3.3)
MODIFIED	3	This parameter selects a set of modified values from the history database by specifying a time domain for one or more *Variables*. A modified value is a value that has been replaced by another value at the same timestamp in the history database. If there are multiple replaced values the server must return all of them. The server indicates that modified data exists by setting the *ExtraData* bit in the *StatusCode* associated with a *DataValues* returned during a RAW, PROCESSED, or ATTIME request.	ReadRawModifiedDetails (See Clause 5.3.3)
PROCESSED	4	This parameter selects a set of aggregate values from the history database by specifying a time domain for one or more *Variables*. This function is intended to provide Values calculated with respect to the resample interval. For example, this function can provide hourly statistics such as Maximum, Minimum, Average, etc. for each item during the specified time domain when resample interval is 1 hour.	ReadProcessedDetails (See Clause 5.3.4)
ATTIME	5	This parameter selects a set of raw or interpolated values from the history database by specifying a series of timestamps for one or more *Variables*. This function is intended to provide values to correlate with other values with a known timestamp. For example, the values of sensors when lab samples were collected.	ReadAtTimeDetails (See Clause 5.3.5)

5.3.2 ReadEventDetails structure

Table 17 defines the *ReadEventDetails* structure. Two of the three parameters, numValuesPerNode, startTime, and endTime must be specified.

Table 17 – 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 must 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.Min* indicates that there is no start time.
endTime	UtcTime	End of period to read. The default value of *DateTime.Min* indicates that there is no end time.
filter	EventFilter	A filter used by the *Server* to determine which HistoricalEventNode should be included. This parameter must 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 [UA Part 4]. This filter also specifies the *EventFields* that are to be returned as part of the request.

The EVENTS parameter reads 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 [UA Part 4], in particular *MonitoredItems*.

The time domain of the request is defined by startTime, endTime, and numValuesPerNode; at least two of these must be specified. If endTime is less than startTime, or endTime and numValuesPerNode alone are specified, the data will be returned in reverse order, with later data coming first, as if time were flowing backward. If all three are specified, 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. The default value is used to indicate when startTime, endTime or numValuesPerNode is 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, time is presumed to be flowing forward. If no data exists at the time specified then the server must return the *Good_NoData StatusCode*.

If more than *numValuesPerNode* results exist within that time range, the *StatusCode* returned for that Variable must be *Good_MoreData,* and the *continuationPoint* must be returned. When

Good_MoreData is returned, clients wanting the next numValuesPerNode values should call HistoryRead again with the continuationPoint.

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* must be *Bad_NoData*.

5.3.3 ReadRawModifiedDetails structure

5.3.3.1 ReadRawModifiedDetails structure Overview

Table 18 defines the *ReadRawDetails* structure. Two of the three parameters, numValuesPerNode, startTime, and endTime must be specified.

Table 18 – ReadRawModifiedDetails

Name	Type	Description
ReadRawModifiedDetails	Structure	Specifies the details used to perform a “raw” or “modified” history read.
isReadModified	Boolean	TRUE for MODIFIED, FALSE for RAW. Default value is FALSE.
startTime	UtcTime	Beginning of period to read. Set to default value of *DateTime.Min* if no specific start time is specified.
endTime	UtcTime	End of period to read. Set to default value of *DateTime.Min* 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 must 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.

5.3.3.2 RAW usage

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 clients wanting the actual data saved within the historian. The actual data may be compressed or may be all 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 clients 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 must be specified. If endTime is less than startTime, or endTime and numValuesPerNode alone are specified, the data will be returned in reverse order, with later data coming first, as if time were flowing backward. If all three are specified, 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.Min* 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, time is presumed to be flowing forward. It is specifically not allowed for the server to return an *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 more than numValuesPerNode results exist within that time range, the *StatusCode* entry for that variable shall be *Good_MoreData*, and the continuationPoint will be set. When *Good_MoreData* is returned, clients 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, 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 reverse order is needed). If numValuesPerNode is 2, the start bound and the first data point is returned (the End bound if reverse order is needed).

When bounding values are requested and no bounding value is found, the corresponding *StatusCode* entry will be set to *Bad_NoBound*, 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, the corresponding *StatusCode* must be *Good_NoData*. If bounding values are requested and one or both exist, 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 must return the most recent value. If the server returns a value which hides other values at a timestamp then it must set the *ExtraData* bit in the *StatusCode* associated with that value.

5.3.3.3 MODIFIED usage

When this structure is used for reading Modified Values (isReadModified is set to true); it reads the values, qualities, timestamps, user identifier, and timestamp of the modification from the history database for the specified time domain for one or more *HistoricalDataNodes*.

The purpose of this function is to read values from history that have been modified/replaced. If ReadRaw, ReadProcessed, or ReadAtTime has returned a *StatusCode* of with the *ExtraData* bit set then there are values which have been superseded in the history database. This parameter allows clients to read those values which were superseded. Only values that have been modified/replaced or deleted are read by this function

The domain of the request is defined by startTime, endTime, and numValuesPerNode; at least two of these must be specified. If endTime is less than startTime, or endTime and numValuesPerNode alone are specified, the data shall be returned in reverse order, with later data coming first. If all three are specified, 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 results exist within that time range, the *StatusCode* entry for that variable shall be *Good_MoreData*. If numValuesPerNode is 0, then all the values in the range are returned.

If a value has been modified multiple times, 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 most recent to oldest modified value, if startTime is less than or equal to endTime. If endTime is less than startTime, the order of the returned values will be from oldest modified value to most recent. It is server dependent whether multiple modifications are kept or only the most recent.

5.3.4 ReadProcessedDetails structure

Table 19 defines the structure of the ReadProcessedDetails structure.

Table 19 – 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.
resampleInterval	Duration	Interval between returned aggregate values. The value 0 indicates that there is no interval defined.
aggregateType	NodeId	The NodeId of the HistoryAggregate object that indicates the aggregate to be used when retrieving processed history. See for details.

See Table 8 for possible *NodeId* values for the HistoryAggregateType parameter.

The PROCESSED function computes 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 subintervals of duration resampleInterval. The specified aggregateType is calculated for each subinterval beginning with startTime by using the data within the next resampleInterval.

For example, this function can provide hourly statistics such as Maximum, Minimum, Average, etc. for each item during the specified time domain when resampleInterval is 1 hour.

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

The values used in computing the aggregate for each subinterval shall include any value that falls exactly on the timestamp beginning the subinterval, but shall not include any value that falls directly on the timestamp ending the subinterval. Thus, each value shall be included only once in the calculation. If the time domain is in reverse order, 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 subintervals being requested in the two cases are not the same.

If the last subinterval computed is not a complete subinterval (the time domain of the request is not evenly divisible by the resample interval), the last aggregate returned shall be based upon that incomplete subinterval, and the corresponding *StatusCode* shall be *PARTIAL*.

For MinimumActualTime and MaximumActualTime, if more than one instance of the value exists within a subinterval, which instance (time stamp) of the value returned is server dependent. In any case, the server must set the *MultipleValue* bit in the *StatusCode* to let the caller know that there are other timestamps with that value.

If resampleInterval is 0, the server must create one aggregate value for the entire time range. This allows aggregates over large periods of time. A value with a timestamp equal to endTime will be excluded from that aggregate, just as it would be excluded from a subinterval with that ending time.

The timestamp returned with the aggregate must be the time at the beginning of the interval, except where the aggregate specifies a different value.

For all Aggregates that do not specify otherwise the following rule applies to determining the status associated with a given computed value. If the percentage of the values used in computing the aggregate value that have *Good* quality meets or exceeds the PercentDataGood parameter, the *StatusCode* of the aggregate must be *Good*. If the percentage of the values used in computing the aggregate value that have *Bad* quality meets or exceeds the PercentDataBad parameter, the *StatusCode* of the aggregate must be *Bad*. Otherwise the *StatusCode* of the aggregate must be *Uncertain_SubNormal*.

If no data exists for a given *HistoricalDataNode* in any subinterval in the time domain, the server shall return *Bad_NoData* in the *StatusCode* for that *HistoricalDataNode*.

If data does exist in at least one subinterval for that *HistoricalDataNode*, the server shall return a timestamp, StatusCode, and value for each subinterval in the time domain.

5.3.5 ReadAtTimeDetails structure

Table 20 defines the ReadAtTimeDetails structure.

Table 20 – 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.

The ATTIME parameter 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 time stamps 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 Clause 5.6.3.5

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

5.4 HistoryData parameters

5.4.1 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 [UA Part 4] for details on *Extensible Parameters*.

5.4.2 HistoryData type

Table 21 defines the structure of the *HistoryData* used for the data to return in a *HistoryRead*.

Table 21 – HistoryData Details

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

5.4.3 HistoryEvent type

Table 22 – HistoryEvent Details

Name	Type	Description
historyEvent[]	EventNotification	An array of Event *Notification* data. The size of the array depends on the requested data parameters.

5.5 HistoryUpdateDetails parameter

5.5.1 Overview

The *HistoryUpdate* service defined in [UA Part 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 [UA Part 4] for the definition of *Extensible Parameter*. Table 23 lists the valid values for the *parameterTypeId* parameter which specifies which function the *HistoryUpdate* service will perform, and what structure will be contained in the *parameterData* field .

Table 23 – HistoryUpdateDetails parameterTypeId Values

Name	Value	Description	parameterData Structure
INSERTDATA	1	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 (See Clause 5.5.2)
REPLACEDATA	2	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 (See Clause 5.5.2)
UPDATEDATA	3	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.	UpdateDataDetails (See Clause 5.5.2)
INSERTEVENT	4	This function inserts new events into the history database for one or more HistoricalEventNodes.	UpdateEventDetails (See Clause 5.5.3)
REPLACEEVENT	5	This function replaces values of fields in existing events into the history database for one or more HistoricalEventNodes.	UpdateEventDetails (See Clause 5.5.3)
UPDATEEVENT	6	This function inserts new events or replaces values of fields in existing events into the history database for one or more HistoricalEventNodes.	UpdateEventDetails (See Clause 5.5.3)
DELETERAW	7	This function deletes all values from the history database for the specified time domain for one or more HistoricalDataNodes.	DeleteDataDetails (See Clause 5.5.4)
DELETEMODIFIED	8	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. .	DeleteDataDetails (See Clause 5.5.4)
DELETEATTIME	9	This function deletes all values in the history database for the specified timestamps for one or more HistoricalDataNodes. .	DeleteAtTimeDetails (See Clause 5.5.5)
DELETEEVENT	10	This function deletes events from the history database for the specified filter for one or more HistoricalEventNodes.	DeleteEventDetails (See Clause 5.5.6)

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 is described in [UA Part 4]. This description assumes the user issuing the request and the server that is processing the request supports Updating entries. See [UA Part 3] for a description of *Attributes* that expose the support of Historical Updates.

5.5.2 UpdataDataDetails structure

5.5.2.1 UpdataDataDetails structure Overview

Table 24 defines the UpdateDataDetails structure.

Table 24 – UpdateDataDetails

Name	Type	Description
UpdateDataDetails	Structure	The details for insert, replace, and insert/replace history updates.
performInsert	Boolean	TRUE means perform INSERT, FALSE means do not perform INSERT. Default is FALSE.
performReplace	Boolean	TRUE means perform REPLACE, FALSE means do not perform REPLACE. Default is FALSE.
nodeId	NodeId	Node id of the variable to be updated.
updateValue	historyData	New value to be inserted or replaced

5.5.2.2 INSERTDATA usage

The INSERTDATA parameter inserts entries into the history database at the specified timestamps for one or more *HistoricalDataNodes*. If an entry exists at the specified timestamp, 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.

5.5.2.3 REPLACEDATA usage

The REPLACEDATA parameter replaces entries in the history database at the specified timestamps for one or more *HistoricalDataNodes*. If no entry exists at the specified timestamp, 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.

5.5.2.4 UPDATEDATA usage

The UPDATEDATA parameter inserts or replaces entries in the history database for the specified timestamps for one or more *HistoricalDataNodes*. If the item has a entry at the specified timestamp, the new entry will replace the old one. If there is no entry at that timestamp, the function will insert the new data.

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 a entry that was already there, it should use *Good_EntryInserted* or *Good_EntryReplaced* to return that information.

5.5.3 UpdateEventDetails structure

Table 24 defines the UpdateEventDetails structure.

Table 25 – UpdateEventDetails

Name	Type	Description
UpdateEventDetails	Structure	The details for insert, replace, and insert/replace history event updates.
performInsert	Boolean	TRUE means perform INSERT, FALSE means do not perform INSERT. Default is FALSE.
performReplace	Boolean	TRUE means perform REPLACE, FALSE means do not perform REPLACE. Default is FALSE.
nodeId	NodeId	Node id of the Node to be updated.
filter	EventFilter	If the history of *Notification* conforms to the *EventFilter*, the history of the *Notification* is updated.
eventData	EventNotification	Event *Notification* data to be inserted or updated.

5.5.3.1 INSERTEVENT usage

The INSERTEVENT parameter inserts entries into the event history database for one or more *HistoricalEventNodes*. The *whereClause* parameter of the *EventFilter* must specify the *EventId* Property. If any entry exists matching the specified filter, the new entry shall not be inserted; instead *StatusCode* shall indicate *Bad_EntryExists*.

If the new entry is incomplete or not correctly specified in the *EventNotification*, the server may return a *StatusCode* of *Bad_InvalidArgument*.

This function is intended to insert new entries; e.g., backfilling of historical events.

5.5.3.2 REPLACEEVENT usage

The REPLACEEVENT parameter replaces entries in the event history database for the specified filter for one or more *HistoricalEventNodes*. The *whereClause* parameter of the *EventFilter* must specify the *EventId* Property. If no entry exists matching the specified filter, the new entry shall not be inserted; otherwise the *StatusCode* shall indicate *Bad_NoEntryExists*.

If the new entry is incomplete or not correctly specified in the *EventNotification*, the server may return a *StatusCode* of *Bad_InvalidArgument*.

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.

5.5.3.3 UPDATEEVENT usage

The UPDATEEVENT parameter inserts or replaces entries in the event history database for the specified filter for one or more *HistoricalEventNodes*. The *whereClause* parameter of the *EventFilter* must specify fields to uniquely identify the event (i.e. EventId or combination of identifying fields). If any entry at exists matching the specified filter, the new event data will replace the existing data. If no matching entry is found, the function will insert the new event.

This function is intended to unconditionally insert/replace events; e.g., synchronizing a backup event database.

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

5.5.4 DeleteRawModifiedDetails structure

Table 26 defines the DeleteRawModifiedDetails structure.

Table 26 – DeleteRawModifiedDetails

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

The DELETERAW parameter deletes all raw entries from the history database for the specified time domain for one or more *HistoricalDataNodes*.

The DELETEMODIFIED parameter deletes all modified entries from the history database for the specified time domain for one or more *HistoricalDataNodes*.

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.

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

5.5.5 DeleteAtTimeDetails structure

Table 27 defines the structure of the DeleteAtTimeDetails structure.

Table 27 – DeleteAtTimeDetails

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

The DELETEATTIME parameter 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.

5.5.6 DeleteEventDetails structure

Table 27 defines the structure of the DeleteEventDetails structure.

Table 28 – DeleteEventDetails

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

The DELETEEVENT parameter 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*, the *StatusCode* for that *Node* is *Bad_NoData.*

5.6 Aggregate Details

5.6.1 General

The purpose of this section is to detail the requirements and behavior for OPC UA Server supporting Historical Access *Aggregates*. The intent is to standardize the OPC UA Server supporting Historical Access *Aggregates* such that OPC UA Server supporting Historical Access clients can reliably predict the results of an *Aggregate* computation and understand its meaning. If users require custom functionality in the *Aggregates*, those *Aggregates* should be written as custom vendor defined *Aggregates*.

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

This section is divided up into two parts. The first sub section deals with *Aggregate* characteristics and behavior that are common to all *Aggregates*. The remaining sub sections deal with the characteristics and behavior of *Aggregates* that are aggregate-specific.

5.6.2 Common characteristics

5.6.2.1 Description

This subsection deals with aggregate characteristics and behavior that are common to all aggregates.

5.6.2.2 Generating intervals

To read aggregates, OPC clients must specify three time parameters:

- start time (Start)

- end time (End)

- resample interval (Int)

The OPC server must use these three parameters to generate a sequence of time intervals and then calculate an aggregate for each interval. This section specifies, given the three parameters, which time intervals are generated. Table 29 out lines information on the intervals for each Start and End time combination. Range is defined to be |End - Start|.

All interval aggregates return a timestamp of the start of the interval unless otherwise noted for the particular aggregate.

Table 29 – History Aggregate Interval Information

Start/End Time	Resample Interval	Resulting Intervals
Start = *End*	*Int* = Anything	No intervals. Returns a Bad_InvalidArgument *StatusCode*, regardless of whether there is data at the specified time or not.
Start < *End*	*Int* = 0 or *Int* ≥ *Range*	One interval, starting at *Start* and ending at *End*. Includes *Start*, excludes *End*, i.e., *[Start, End)*.
Start < *End*	*Int* ≠ 0, *Int* < *Range, Int* divides *Range* evenly.	*Range/Int* intervals. Intervals are *[Start, Start + Int), [Start + Int, Start + 2 * Int),..., [End - Int, End)*.
Start < *End*	*Int* ≠ 0, *Int* < *Range, Int* does not divide *Range* evenly.	⌈*Range/Int*⌉ intervals. Intervals are *[Start, Start + Int), [Start + Int, Start + 2 * Int),..., [Start + (⌊Range/Int⌋ - 1)* Int, Start + ⌊Range/Int⌋ * Int), [Start + ⌊Range/Int⌋ * Int, End)*. In other words, the last interval contains the "rest" that remains in the range after taking away ⌊*Range/Int*⌋ intervals of size *Int*.
Start > *End*	*Int* = 0 or *Int* ≥ *Range*	One interval, starting at *Start* and ending at *End*. Includes *Start*, excludes *End*, i.e., *[End, Start)*.
Start > *End*	*Int* ≠ 0, *Int* < *Range, Int* divides *Range* evenly.	*Range/Int* intervals. Intervals are *[Start - Int, Start], [Start – 2 * Int, Start – Int),..., [End, End + Int)*.
Start > *End*	*Int* ≠ 0, *Int* < *Range, Int* does not divide *Range* evenly.	⌈*Range/Int*⌉ intervals. Intervals are *[Start - Int, Start), [Start – 2 * Int, Start - Int),..., [Start - ⌊Range/Int⌋ * Int, Start – (⌊Range/Int⌋ - 1)* Int), [End, Start - ⌊Range/Int⌋ * Int)*. In other words, the last interval contains the "rest" that remains in the range after taking away ⌊*Range/Int*⌋ intervals of size *Int* starting at Start.

5.6.2.3 Data types

Table 8 outlines the valid data types for each aggregate. Some aggregates are intended for numeric data types – i.e. integers or real/floating point numbers. Dates, strings, arrays, etc. are not supported. Other aggregates are intended for digital data types – i.e. Boolean or enumerations. In addition some aggregates may return results with a different datatype than those used to calculate the aggregate. Table 8 also outlines the default data type returned for each aggregate.

Table 30 – Standard History Aggregate Data Type Information

BrowseName	Valid Data Type	Default Result Data Type
	Interpolation Aggregate	
Interpolative	Numeric	Double
	Data Averaging Aggregates	
Average	Numeric	Double
TimeAverage	Numeric	Double
Total	Numeric	Double
TotalizeAverage	Numeric	Double
	Data Variation Aggregates	
Minimum	Numeric	Raw data type
Maximum	Numeric	Raw data type
MinimumActualTime	Numeric	Raw data type
MaximumActualTime	Numeric	Raw data type
Range	Numeric	Raw data type
	Counting Aggregates	
AnnotationCount	All	Integer
Count	All	Integer
DurationInState0	Boolean	Duration
DurationInState1	Boolean	Duration
NumberOfTransitions	Boolean	Integer
	Time Aggregates	
Start	All	Raw data type
End	All	Raw data type
Delta	Numeric	Raw data type
	Data Quality Aggregates	
DurationGood	All	Duration
DurationBad	All	Duration
PercentGood	All	Double
PercentBad	All	Double
WorstQuality	Numeric	StatusCode

5.6.2.4 StatusCode calculation

For Aggregate values, the *StatusCode* for each returned aggregate shall be *Good*, if the *StatusCode* for ALL values used in the aggregate was *Good*.

If the *StatusCode* of ANY value used in computing the aggregate was not *Good*, then the server must use the *TreatUncertainAsBad*, *PercentDataBad* and *PercentDataGood* parameter (see Clause 4.8.2) settings to determine the *StatusCode* of the resulting aggregate for the interval. Some aggregates may explicitly define there own method of determining quality.

If the percentage of *Good* values in an interval is greater then or equal to the *PercentDataGood*, the aggregate is considered *Good*.

If the percentage of *Bad* values in an interval is greater then or equal to the *PercentDataBad*, the aggregate is considered *Bad*.

Since a value can be either *Good* or *Bad* only (*Uncertain* is defined as *Good* or *Bad* as per *TreatUncertainAsBad* setting), percentage good = 100 – percentage bad. If a percentage good (X) is in the following range Percentage bad < X < Percentage Good then the quality of the aggregate is *Uncertain_SubNormal*.

5.6.3 Aggregate specific characteristics

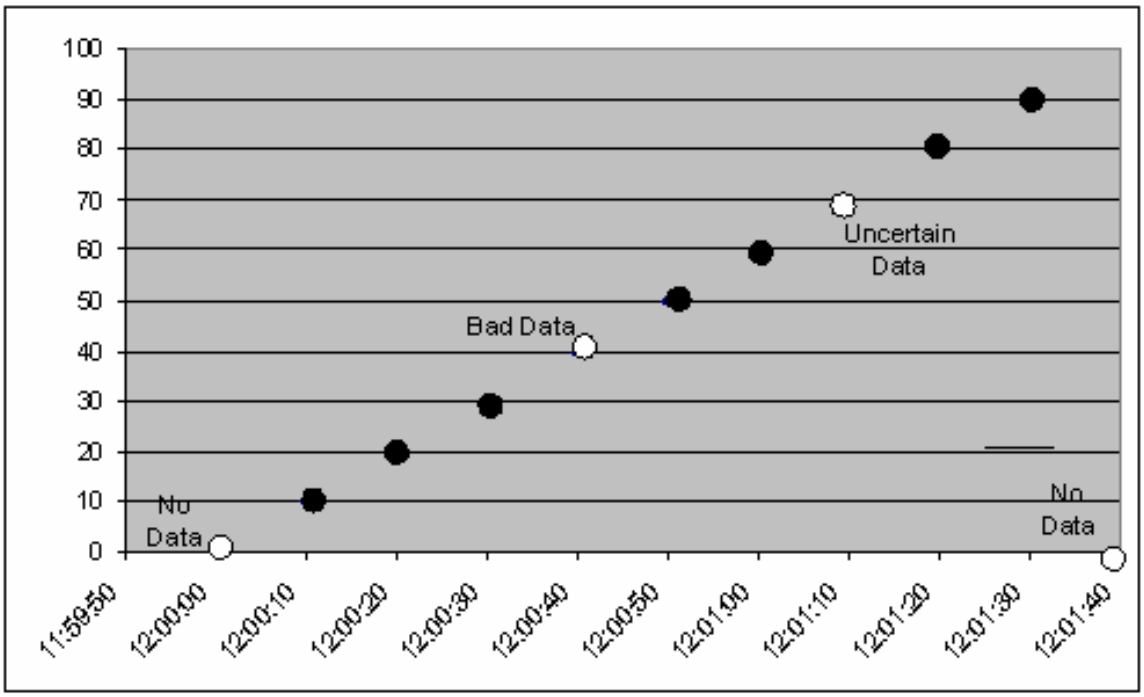
5.6.3.1 Description

This sub section deals with aggregate specific characteristics and behavior that is specific to a particular aggregate.

5.6.3.2 Example aggregate data – Historian 1

For the purposes of examples consider a source historian with the following data:

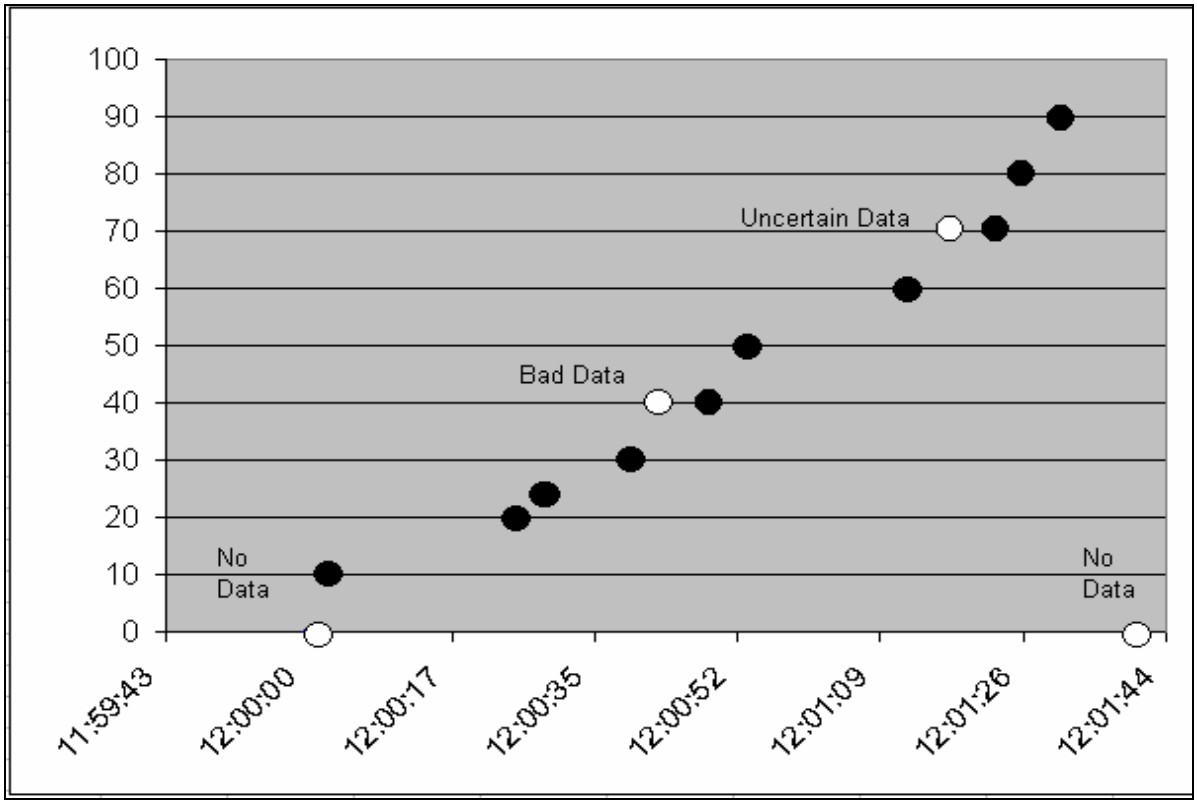
Timestamp	Value	StatusCode	Notes
Jan-01-2002 12:00:00	-	Bad_NoData	First archive entry, Point Created
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:20	20	Raw, Good	
Jan-01-2002 12:00:30	30	Raw, Good	
Jan-01-2002 12:00:40	40	Raw, Bad	Scan failed, Bad data entered
Jan-01-2002 12:00:50	50	Raw, Good	
Jan-01-2002 12:01:00	60	Raw, Good	
Jan-01-2002 12:01:10	70	Raw, Uncertain	Value is flagged as questionable
Jan-01-2002 12:01:20	80	Raw, Good	
Jan-01-2002 12:01:30	90	Raw, Good	
	NULL	No Data	No more entries, awaiting next scan.



5.6.3.3 Example aggregate data – Historian 2

The following data is also included in a separate column to illustrate non-periodic data

Timestamp	Value	StatusCode	Notes
Jan-01-2002 12:00:00	-	Bad_NoData	First archive entry, Point Created
Jan-01-2002 12:00:02	10	Raw, Good	
Jan-01-2002 12:00:25	20	Raw, Good	
Jan-01-2002 12:00:28	25	Raw, Good	
Jan-01-2002 12:00:39	30	Raw, Good	
Jan-01-2002 12:00:42	40	Raw, Bad	Bad quality data received, Bad data entered
Jan-01-2002 12:00:48	40	Raw, Good	Received good StatusCode value
Jan-01-2002 12:00:52	50	Raw, Good	
Jan-01-2002 12:01:12	60	Raw, Good	
Jan-01-2002 12:01:17	70	Raw, Uncertain	Value is flagged as questionable
Jan-01-2002 12:01:23	70	Raw, Good	
Jan-01-2002 12:01:26	80	Raw, Good	
Jan-01-2002 12:01:30	90	Raw, Good	
	-	No Data	No more entries, awaiting next Value.



5.6.3.4 Example Conditions

For the purposes of all examples,

Historian 1

1. *TreatUncertainAsBad* = False. Therefore *Uncertain* values are included in aggregate call.

2. *Stepped* attribute = False. Therefore Linear interpolation is used between data points.

3. *SteppedInterpolationMode* = True. Therefore Stepped extrapolation is used at end boundary conditions

Historian 2

1. *TreatUncertainAsBad* = True. Therefore *Uncertain* values are treated as *Bad*, and not included in the aggregate call.

2. *Stepped* attribute = False. Therefore Linear interpolation is used between data points.

3. *SteppedInterpolationMode* = True, Therefore Stepped extrapolation is used at end boundary conditions

5.6.3.5 Interpolative

5.6.3.5.1 Description

In order for the interpolative aggregate to return meaningful data, there must be good values at the boundary conditions. For the purposes of discussion we will use the terms good and non-good. As discussed in the *StatusCode* section (See Clause 5.6.2.4), what is represented by non-good is Server dependant. For some Servers non-good represents only *Bad* data, for others it represents *Bad* and *Uncertain* data depending on the *TreatUncertainAsBad* setting.

When determining boundary conditions, the following rules must be followed:

- If the value at the requested time is non-good, the aggregate looks for good bounding data within the intervals preceding and following the requested time. (In the case of Stepped interpolation a bounding value following requested time is not required). If no good data is found within the respective intervals, there is no bound and the aggregate must return *Bad_NoData.* If no data exists within the respective intervals, the aggregate will continue expanding the respective search intervals up to a maximum equal to the requested time Range. If no data exists within the respective Ranges, there is no bound and the aggregate must return *Bad_NoData.*

- The method of interpolation, either interpolated (sloped Lines between point) or as Stepped (vertically-connected horizontal lines between points) is determined by the *Stepped* attribute. See Clause 4.7.1.4

- If there is no end bound (i.e. future time), the value should be extrapolated forward in time from the previous good value. The method of extrapolation, Stepped (i.e. hold last value) or extrapolated (extend line based on preceding slope) will be server dependant. This is indicated by the *SteppedInterpolationMode* property. See Clause 4.8.2.

- The aggregate should not extrapolate backwards in time. If there is no beginning bound, it must return *Bad_NoData*. The trailing value should not be pulled backward in time.

- If there happens to be a good raw value at the requested time, the raw value is returned.

- If any non-good values are skipped in order to find the closest good value, the aggregate will be *Uncertain_Subnormal*

- Unless otherwise indicated, *StatusCodes* are *Good, Interpolated*.

The following examples demonstrate the various situations:

5.6.3.5.2 Interpolated data with good bounding value.

Start: Jan-01-2002 12:00:10 **End:** Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:00:10	10	Raw, Good	13.5	Interpolated, Good	Value2 –Interpolated between values at 12:00:02 and 12:00:25
Jan-01-02 12:00:15	15	Interpolated, Good	15.7	Interpolated, Good	Value2 –Interpolated between values at 12:00:02 and 12:00:25

5.6.3.5.3 Interpolated data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End:** Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:00:35	35	Interpolated, Uncertain	28.2	Interpolated, Good	Value2 –Interpolated between values at 12:00:28 and 12:00:39
Jan-01-02 12:00:40	40	Interpolated, Uncertain	31.1	Interpolated, Uncertain	Raw Value is Bad, Value2 – Interpolated between values at 12:00:39 and 12:00:48
Jan-01-02 12:00:45	45	Interpolated, Uncertain	36.7	Interpolated, Uncertain	Bounding Value Bad, Value2 – Interpolated between values at 12:00:39 and 12:00:48
Jan-01-02 12:00:50	50	Raw, Good	45	Interpolated, Good	
Jan-01-02 12:00:55	55	Interpolated, Good	51.5	Interpolated, Good	

5.6.3.5.4 Interpolated data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End:** Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:01:20	80	Raw, Good	67.3*	Interpolated, Uncertain	Uncertain Values excluded. Value2 –Interpolated between values at 12:01:12 and 12:01:23
Jan-01-02 12:01:25	85	Interpolated, Good	76.7	Interpolated, Good	
Jan-01-02 12:01:30	90	Raw, Good	90	Raw, Good	
Jan-01-02 12:01:35	90	Interpolated, Uncertain	90	Interpolated, Uncertain	Bounding Value at 12:01:30, Extrapolated using stepped method

* If Historian 2 had treated *Uncertain* values as *Good*. The value would be 70, interpolated between 12:00:17 and 12:00:2323 and the quality would be "*Interpolated,Good*".

5.6.3.5.5 Interpolated data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End:** Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	0	No Data, Bad	No bounding Value, do not extrapolate
Jan-01-2002 12:00:05	-	No Data, Bad	11.3	Interpolated, Good	Value 1 - No bounding value, do not extrapolate Value2 –Interpolated between values at 12:00:02 and 12:00:25
Jan-01-2002 12:00:10	10	Raw, Good	13.5	Interpolated, Good	Value2 –Interpolated between values at 12:00:02 and 12:00:25
Jan-01-2002 12:00:15	15	Interpolated, Good	15.7	Interpolated, Good	Value2 –Interpolated between values at 12:00:02 and 12:00:25

5.6.3.6 Average

5.6.3.6.1 Description

The average aggregate adds up the values of all good raw data for each interval, and divides the sum by the number of good values. If any non-good values are ignored in the computation, the aggregate *StatusCode* will be determined using the StatusCode Calculation (See Clause 5.6.2.4)

If no data exists for an interval, the *StatusCode* of the aggregate for that interval will be *Good_NoData*.

All interval aggregates return timestamp of the start of the interval. Unless otherwise indicated, *StatusCodes* are *Good, Calculated*.

5.6.3.6.2 Average data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	10	Calculated, Good	-	No Data, Bad	Value2-No Raw data in interval
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	No Raw data in intervals

5.6.3.6.3 Average data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	30	Calculated, Good	
Jan-01-2002 12:00:40	-	No Data, Bad	-	No Data, Bad	Value1-Only Bad data in interval Value 2- No data in interval
Jan-01-2002 12:00:45	-	No Data, Bad	40	Calculated, Good	Value 1- No data in interval
Jan-01-2002 12:00:50	50	Calculated, Good	50	Calculated, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	-	No Data, Bad	No data in intervals

5.6.3.6.4 Average data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:20	80	Calculated, Good	70	Calculated, Good	
Jan-01-2002 12:01:25	-	No Data, Bad	80	Calculated, Good	Value 1- No data in interval
Jan-01-2002 12:01:30	90	Calculated, Good	90	Calculated, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	-	No Data, Bad	No data in intervals

5.6.3.6.5 Average data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	10	Partial, Good	Value1- No data in interval Value2 - Partial interval :02-:05
Jan-01-2002 12:00:05	-	No Data, Bad	-	No Data, Bad	No data in intervals
Jan-01-2002 12:00:10	10	Calculated, Good	-	No Data, Bad	Value 2- No data in interval
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	No data in intervals

5.6.3.7 TimeAverage

5.6.3.7.1 Description

The time weighted average aggregate uses interpolation as described in the interpolated section above to find the value of a point at the beginning and end of an interval. A straight line is drawn between each raw value in the interval. The area under the line is divided by the length of the interval to yield the average.

For Example:

Given:

Start: Jan-01-2002 12:00:10

End: Jan-01-2002 12:00:15

Interval: 00:00:05

Then:

Point1 = Good Raw value of 10 at 12:00:10

Point2 = interpolated value of 15 at 12:00:15, using bounding values at 12:00:10 and 12:00:20.

Area under the line is 62.5 (1/2 base*height + base*height). Interval is 5 seconds

TimeAverage = Area/interval = 12.5

If any of an interval's raw values are non-good, they are ignored, and the aggregate *StatusCode* for that interval is determined using the StatusCode Calculation (See Clause 5.6.2.4)

All cases use the interpolated values determined in Cases outlined in section 5.6.3.5 for the bounding values.

5.6.3.7.2 TimeAverage data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	12.5	Calculated, Good	14.5	Calculated, Good	Area under the line between 12:00:10 and 12:00:15 divided by interval length of 5
Jan-01-2002 12:00:15	17.5	Calculated, Good	16.7	Calculated, Good	

5.6.3.7.3 TimeAverage data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	37.5	Calculated, Uncertain	29.7	Calculated, Uncertain	Value1– Interpolate values at :35 and :40 using bounds at :30 and :50 Value2– Interpolate values at :35 and :40 using bounds at :28 and :48 Uncertain means Bad Value ignored
Jan-01-2002 12:00:40	42.5	Calculated, Uncertain	33.9	Calculated Uncertain	Value1– Interpolate values at :40 and :45 using bounds at :30 and :50 Value2– Interpolate values at :40 and :45 using bounds at :39 and :48 Uncertain means Bad Value ignored
Jan-01-2002 12:00:45	47.5	Calculated, Uncertain	40.9	Calculated Uncertain	Value1– Interpolate value at :45 using bounds at :30 and :50 Value2– Interpolate value at :45 using bounds at :39 and :48 Interpolate Value at :50 using bounds at :48 and :52 Uncertain means Bad Value ignored
Jan-01-2002 12:00:50	52.5	Calculated, Good	48.3	Calculated, Good	Value1– Interpolate value at :55 using bounds at :50 and 01:00 Value2– Interpolate value at :50 using bounds at :48 and :52 Interpolate Value at :55 using bounds at :52 and :01:12
Jan-01-2002 12:00:55	57.5	Calculated, Good	52.8	Calculated, Good	Value1– Interpolate value at :55 using bounds at :50 and 01:00 Value2– Interpolate value at :50 using bounds at :48 and :52 Interpolate Value at :55 using bounds at :52 and :01:12

5.6.3.7.4 TimeAverage data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:20	82.5	Calculated, Good	72.0	Calculated Uncertain	Value1– Interpolate value at :25 using bounds at :20 and :30 Value2– Interpolate value at :20 using bounds at :16 and :23 (Uncertain value at :17 is ignored by this historian) Interpolate Value at :25 using bounds at :23 and :26
Jan-01-2002 12:01:25	87.5	Calculated, Good	83.3	Calculated, Good	Value1– Interpolate value at :25 using bounds at :20 and :30 Value2– Interpolate value at :25 using bounds at :23 and :26
Jan-01-2002 12:01:30	90*	Calculated, Uncertain	90*	Calculated, Uncertain	Extrapolate Value at :35 using value at :30
Jan-01-2002 12:01:35	90*	Calculated, Uncertain	90*	Calculated, Uncertain	Extrapolate Values at :35 and :40 using value at :30

* Stepped extrapolation is used at the boundary. Servers may opt to extrapolate data based on the previous slope.

5.6.3.7.5 TimeAverage data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	10.7	Partial, Uncertain	Value1-No bounding value, do not extrapolate. No data in the interval Value2- Interpolate value at :05 using bounds at :02 and :25 Use partial interval :02 to :05, with interval of 3.
Jan-01-2002 12:00:05	-	No Data, Bad	12.4	Calculated, Good	Value1-No bounding value, do not extrapolate. No data in the interval Value2- Interpolate values at :05 and 10 using bounds at :02 and :25
Jan-01-2002 12:00:10	12.5	Calculated, Good	14.5	Calculated, Good	Value1– Interpolate value at :15 using bounds at :10 and :20 Value2– Interpolate values at :10 and :15 using bounds at :02 and :25
Jan-01-2002 12:00:15	17.5	Calculated, Good	16.7	Calculated, Good	Value1– Interpolate value at :15 using bounds at :10 and :20 Value2– Interpolate values at :15 and :20 using bounds at :02 and :25

5.6.3.8 Total

5.6.3.8.1 Description

The total aggregate adds up all the values of all good raw values for each interval. If any non-good values are ignored in the computation, the aggregate *StatusCode* will be determined using the StatusCode Calculation (See Clause 5.6.2.4).

If no data exists for an interval, the *StatusCode* of the aggregate for that interval will be *Good_NoData*.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.8.2 Total data with good bounding value.

Start: Jan-01-2002 12:00:10 **End:** Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:00:10	10	Raw, Good	10	Calculated, Good	
Jan-01-02 12:00:15	0	Calculated, Good	0	Calculated, Good	

5.6.3.8.3 Total data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End:** Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:00:35	0	Calculated, Good	30	Calculated, Good	
Jan-01-02 12:00:40	-	No Data, Bad	-	No Data, Bad	
Jan-01-02 12:00:45	-	No Data, Bad	40	Calculated, Good	
Jan-01-02 12:00:50	50	Calculated, Good	50	Calculated, Good	
Jan-01-02 12:00:55	0	Calculated, Good	0	Calculated, Good	

5.6.3.8.4 Total data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End:** Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-02 12:01:20	80	Calculated, Good	70	Calculated, Good	
Jan-01-02 12:01:25	0	Calculated, Good	80	Calculated, Good	
Jan-01-02 12:01:30	90	Calculated, Good	90	Calculated, Good	
Jan-01-02 12:01:35	-	No Data, Bad	-	No Data, Bad	

5.6.3.9 TotalizeAverage

5.6.3.9.1 Description

The TotalizeAverage aggregate performs the following calculation for each interval:

TotalizeAverage = time_weighted_avg * interval_length (sec)

Where:

Time_weighted_avg is the result from the TimeAverage aggregate, using the interval supplied to the TotalizeAverage call.

Interval_length is the interval of the aggregate.

The resulting units would be normalized to seconds, i.e. [time_weighted_avg Units]*sec.

If any non-good values are ignored in the computation of an interval, the aggregate *StatusCode* will be determined using the StatusCode Calculation (See Clause 5.6.2.4).

All interval aggregates return timestamp of the start of the interval. Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.9.2 TotalizeAverage data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	12.5	Calculated, Good	72.5	Calculated, Good	Area under the line between 12:00:10 and 12:00:15
Jan-01-2002 12:00:15	17.5	Calculated, Good	83.5	Calculated, Good	

5.6.3.10 Minimum

5.6.3.10.1 Description

The minimum aggregate is the same as the minimum actual time, except the timestamp of the aggregate will always be the start of the interval for every interval.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*.

5.6.3.10.2 Minimum data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	

5.6.3.10.3 Minimum data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	30	Calculated, Good	
Jan-01-2002 12:00:40	-	No Data, Bad	40	Calculated, Bad	Value1- Only Bad data in interval.
Jan-01-2002 12:00:45	-	No Data, Bad	40	Calculated, Good	
Jan-01-2002 12:00:50	50	Raw, Good	50	Calculated, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	-	No Data, Bad	

5.6.3.10.4 Minimum data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:20	80	Raw, Good	70	Calculated Good	
Jan-01-2002 12:01:25	-	No Data, Bad	80	Calculated Good	
Jan-01-2002 12:01:30	90	Raw, Good	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	-	No Data, Bad	

5.6.3.10.5 Minimum data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	10	Calculated Good	
Jan-01-2002 12:00:05	-	No Data, Bad	-	No Data, Bad	
Jan-01-2002 12:00:10	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	

5.6.3.10.6 Minimum data with Partial Interval.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:05	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:21	30	Partial, Good	20	Partial, Good	

5.6.3.11 Maximum

5.6.3.11.1 Description

This aggregate is the same as the minimum, except the value is the maximum raw value within the interval [s,e).

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*.

5.6.3.11.2 Maximum data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	

5.6.3.11.3 Maximum data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	30	Calculated, Good	
Jan-01-2002 12:00:40	-	No Data, Bad	40	Calculated, Bad	Only Bad data in interval.
Jan-01-2002 12:00:45	-	No Data, Bad	40	Calculated, Good	
Jan-01-2002 12:00:50	50	Raw, Good	50	Calculated, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	-	No Data, Bad	

5.6.3.11.4 Maximum data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:20	80	Raw, Good	70	Calculated Good	
Jan-01-2002 12:01:25	-	No Data, Bad	80	Calculated Good	
Jan-01-2002 12:01:30	90	Raw, Good	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	-	No Data, Bad	

5.6.3.11.5 Maximum data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	10	Calculated Good	
Jan-01-2002 12:00:05	-	No Data, Bad	-	No Data, Bad	
Jan-01-2002 12:00:10	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	-	No Data, Bad	

5.6.3.11.6 Maximum data with Partial Interval.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:05	10	Raw, Good	-	No Data, Bad	
Jan-01-2002 12:00:21	30	Partial, Good	25	Partial, Good	

5.6.3.12 MininumActualTime

5.6.3.12.1 Description

The minimum actual time aggregate retrieves the minimum good raw value within the interval [s,e), and returns that value with the timestamp at which that value occurs. Note that if the same minimum exists at more than one timestamp, the oldest one is retrieved, and the *StatusCode* is set to *MultiValues*. If a non-good value is lower than the good minimum, the *StatusCode* of the aggregate will be determined using the StatusCode Calculation (See Clause 5.6.2.4).

Unless otherwise indicated, *StatusCodes* are *Good,Raw*. If no values are in the interval no data is returned with a timestamp of the start of the interval. If only bad quality values are

available then the status is returned as Bad, Raw, The value is indeterminate, since some system may save bad values, but other may not.

5.6.3.12.2 MininumActualTime data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:15	-	No Data, Bad	No raw data in interval, do not interpolate

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:15	-	No Data, Bad	No raw data in interval, do not interpolate

5.6.3.12.3 MininumActualTime data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:40	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:45	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:50	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	No raw data in interval, do not interpolate

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:39	30	Raw, Good	
Jan-01-2002 12:00:42	-	Raw, Bad	Only Bad data in interval
Jan-01-2002 12:00:48	40	Raw, Good	
Jan-01-2002 12:00:52	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	No raw data in interval, do not interpolate

5.6.3.12.4 MininumActualTime data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:01:20	80	Raw, Good	
Jan-01-2002 12:01:25	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:01:30	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:01:23	70	Raw, Good	
Jan-01-2002 12:01:26	80	Raw, Good	
Jan-01-2002 12:01:30	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	

5.6.3.12.5 MininumActualTime data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:15	-	No Data, Bad	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:02	10	Raw, Good	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:10	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	

5.6.3.12.6 MininumActualTime with Partial Interval.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:30	30	Partial, Good	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:25	20	Partial, Good	

5.6.3.13 MaximumActualTime

5.6.3.13.1 Description

This is the same as the minimum actual time aggregate, except that the value is the maximum raw value within the interval [s,e). Note that if the same maximum exists at more than one timestamp, the oldest one is retrieved, and the *StatusCode* is set to *MultiValues*

Unless otherwise indicated, *StatusCodes* are *Good, Raw*.

5.6.3.13.2 MaximumActualTime data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:15	-	No Data, Bad	No raw data in interval, do not interpolate

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:15	-	No Data, Bad	No raw data in interval, do not interpolate

5.6.3.13.3 MaximumActualTime data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:40	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:45	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:00:50	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	No raw data in interval, do not interpolate

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:39	30	Raw, Good	
Jan-01-2002 12:00:42	-	No Data, Bad	Only Bad data in interval
Jan-01-2002 12:00:48	40	Raw, Good	
Jan-01-2002 12:00:52	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	No raw data in interval, do not interpolate

5.6.3.13.4 MaximumActualTime data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:01:20	80	Raw, Good	
Jan-01-2002 12:01:25	-	No Data, Bad	No raw data in interval, do not interpolate
Jan-01-2002 12:01:30	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:01:23	70	Raw, Good	
Jan-01-2002 12:01:26	80	Raw, Good	
Jan-01-2002 12:01:30	90	Raw, Good	
Jan-01-2002 12:01:35	-	No Data, Bad	

5.6.3.13.5 MaximumActualTime data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:00	-	No Data, Bad	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:15	-	No Data, Bad	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:02	10	Raw, Good	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:10	-	No Data, Bad	
Jan-01-2002 12:00:15	-	No Data, Bad	

5.6.3.13.6 MaximumActualTime with Partial Interval.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:20	20	Raw, Good	
Jan-01-2002 12:00:30	30	Partial, Good	

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:05	-	No Data, Bad	
Jan-01-2002 12:00:28	25	Partial, Good	

5.6.3.14 Range

5.6.3.14.1 Description

The range aggregate finds the difference between the raw maximum and raw minimum values in the interval. If only one value exists in the interval, the range is zero. Note that the range is always zero or positive.

If there are any non-good raw values in the interval, they are ignored, and the aggregate *StatusCode* will be *Uncertain_Subnormal*.

All interval aggregates are returned with timestamp of the start of the interval. Unless otherwise indicated, *StatusCodes* are *Good, Calculated*.

5.6.3.15 AnnotationCount

5.6.3.15.1 Description

This aggregate returns a count of all annotations.

5.6.3.16 Count

5.6.3.16.1 Description

This aggregate retrieves a count of all the raw values within an interval. If one or more raw values are non-good, they are not included in the count, and the aggregate *StatusCode* is determined using the StatusCode Calculation (See Clause 5.6.2.4). If no good data exists for an interval, the count is zero.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.16.2 Count data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	1	Calculated, Good	0	Calculated, Good	
Jan-01-2002 12:00:15	0	Calculated, Good	0	Calculated, Good	

5.6.3.16.3 Count data with uncertain data in the interval.

Start: Jan-01-2002 12:00:50 **End**: Jan-01-2002 12:01:30 **Interval**: 00:00:00

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:50	4*	Calculated, Good	4	Calculated, Uncertain	Value2 treats uncertain as bad, which also changes StatusCode

* For servers with *TreatUncertainAsBad* = True then the result would be 3.

5.6.3.17 DurationInState0

5.6.3.17.1 Description

This aggregate returns the time duration during the resample interval that the variable was in the zero state. If one or more raw values are non-good, they are not included in the duration, and the aggregate *StatusCode* is determined using the StatusCode Calculation (See Clause 5.6.2.4). If no good data exists for an interval, the duration is 0.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.18 DurationInState1

5.6.3.18.1 Description

This aggregate returns the time duration during the resample interval that the variable was in the one state. If one or more raw values are non-good, they are not included in the duration, and the aggregate *StatusCode* is determined using the StatusCode Calculation (See Clause 5.6.2.4). If no good data exists for an interval, the duration is 0.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.19 NumberOfTransitions

5.6.3.19.1 Description

This aggregate returns a count of the number of transition the variable had during the resample interval. If one or more raw values are non-good, they are not included in the duration, and the aggregate *StatusCode* is determined using the StatusCode Calculation (See Clause 5.6.2.4). If no good data exists for an interval, the number of transistions is 0.

Unless otherwise indicated, *StatusCodes* are *Good, Calculated*

5.6.3.20 Start

5.6.3.20.1 Description

The start aggregate retrieves the first raw value within the interval [s,e), and returns that value with the timestamp at which that value occurs. If the value is non-good , than the *StatusCode* of the aggregate will be *Uncertain_Subnormal*. Unless otherwise indicated, *StatusCodes* are *Good, Raw*.

5.6.3.20.2 Start data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	
Jan-01-2002 12:00:15	-	No Data, Bad	Return Timestamp of the interval

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	-	No Data, Bad	Return Timestamp of the interval
Jan-01-2002 12:00:15	-	No Data, Bad	Return Timestamp of the interval

5.6.3.20.3 Start data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	
Jan-01-2002 12:00:40	40	Raw, Bad	Raw Value (If Bad values are stored)
Jan-01-2002 12:00:45	-	No Data, Bad	
Jan-01-2002 12:00:50	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:39	30	Raw, Good	First raw in :35-:40 at :39
Jan-01-2002 12:00:42	40	Raw, Bad	Raw Value (If Bad values are stored)
Jan-01-2002 12:00:48	40	Raw, Good	First raw in :45-:50 at :48
Jan-01-2002 12:00:52	50	Raw, Good	First raw in :50-:55 at :52
Jan-01-2002 12:00:55	-	No Data, Bad	

5.6.3.20.4 Start data with partial intervals.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	First raw in :05-:21 at :10
Jan-01-2002 12:00:30	30	Partial, Good	First raw in :21-:35 at :30

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:25	-	No Data, Bad	No raw data in :05-:21 at :10
Jan-01-2002 12:00:25	20	Raw, Good	First raw in :21-:35 at :25

5.6.3.21 End

5.6.3.21.1 Description

The end aggregate retrieves the last raw value within the interval [s,e), and returns that value with the timestamp at which that value occurs. If the value is non-good , than the *StatusCode* of the aggregate will be *Uncertain_Subnormal*.

Unless otherwise indicated, *StatusCodes* are *Good, Raw*.

5.6.3.21.2 End data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	Last raw in :10-:15 at :10
Jan-01-2002 12:00:15	-	No Data, Bad	Return Timestamp of the interval.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	-	No Data, Bad	Return Timestamp of the interval
Jan-01-2002 12:00:15	-	No Data, Bad	Return Timestamp of the interval

5.6.3.21.3 End data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:35	-	No Data, Bad	
Jan-01-2002 12:00:40	40	Raw, Bad	Raw Value (If Bad values are stored)
Jan-01-2002 12:00:45	-	No Data, Bad	
Jan-01-2002 12:00:50	50	Raw, Good	
Jan-01-2002 12:00:55	-	No Data, Bad	

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:39	30	Raw, Good	Last raw in :35-:40 at :39
Jan-01-2002 12:00:40	40	Raw, Bad	Raw Value (If Bad values are stored)
Jan-01-2002 12:00:48	40	Raw, Good	Last raw in :45-:50 at :48
Jan-01-2002 12:00:52	50	Raw, Good	Last raw in :50-:55 at :52
Jan-01-2002 12:00:55	-	No Data, Bad	

5.6.3.21.4 End data with partial intervals.

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 1		Notes
	Value	StatusCode	
Jan-01-2002 12:00:10	10	Raw, Good	Last raw in :05-:21 at :10
Jan-01-2002 12:00:30	30	Partial, Good	Last raw in :21-:35 at :30

Start: Jan-01-2002 12:00:05 **End**: Jan-01-2002 12:00:35 **Interval**: 00:00:16

Timestamp	Historian 2		Notes
	Value	StatusCode	
Jan-01-2002 12:00:25	-	No Data, Bad	No raw data in :05-:21 at :10
Jan-01-2002 12:00:28	25	Raw, Good	Last raw in :21-:35 at :28

5.6.3.22 Delta

5.6.3.22.1 Description

The delta aggregate retrieves the difference between the earliest and latest good raw values in an interval. If the last value is less than the first value, the result will be negative. If the last value is the same as the first value, or if the last value is also the first value at the same timestamp, the result will be zero. If the last value is greater than the first value, the result will be positive.

If any non-good values exist earlier or later than the earliest and latest good values, respectively, the aggregate is *Uncertain_Subnormal*.

All interval aggregates are returned with timestamp of the start of the interval. Unless otherwise indicated, *StatusCodes* are *Good, Calculated*.

5.6.3.23 DurationGood

5.6.3.23.1 Description

The duration good aggregate looks at the *StatusCode* of a bounding value of the interval to determine what the *StatusCode* is at the beginning of the interval. If no bounding value exists, the *StatusCode* is assumed to be bad at the start of the interval. This aggregate only considers truly Good values. Uncertain values are not considered Good for purposes of calculating this aggregate.

Whenever a raw value x with quality q is encountered from beginning to end within an interval, the quality is considered to be q until the next value, y, is encountered, at which point the quality becomes that of y, and so on.

The time is returned in seconds. No returned value will ever be *Uncertain_Subnormal.*

Each interval's aggregate is returned with timestamp of the start of the interval. *StatusCodes* are *Good, Calculated*

5.6.3.23.2 DurationGood data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	5	Calculated, Good	5	Calculated, Good	
Jan-01-2002 12:00:15	5	Calculated, Good	5	Calculated, Good	

5.6.3.23.3 DurationGood data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	5	Calculated, Good	5	Calculated, Good	Value2-Good from :35 to :39. Good :39 to :40
Jan-01-2002 12:00:40	0	Calculated, Good	2	Calculated, Good	Value2-Good from :40 to :42. Bad :42 to :45
Jan-01-2002 12:00:45	0	Calculated, Good	2	Calculated, Good	Value2-Bad from :45 to :48. Good :48 to :50
Jan-01-2002 12:00:50	5	Calculated, Good	5	Calculated, Good	
Jan-01-2002 12:00:55	5	Calculated, Good	5	Calculated, Good	

5.6.3.23.4 DurationGood data with no good end bounding value.

Start: Jan-01-2002 12:01:20 **End**: Jan-01-2002 12:01:40 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:20	5	Calculated, Good	2	Calculated, Good	Value2-Uncertain from :20 to :23. Good :23 to :25
Jan-01-2002 12:01:25	5	Calculated, Good	5	Calculated, Good	
Jan-01-2002 12:01:30	5	Calculated, Good	5	Calculated, Good	
Jan-01-2002 12:01:35	5	Calculated, Good	5	Calculated, Good	

5.6.3.23.5 DurationGood data with no good start bounding value.

Start: Jan-01-2002 12:00:00 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:00	0	Calculated, Good	3	Calculated, Good	Value1-No bound, Bad from :00 to :05 Value2-Bad from :00 to :02. Good :02 to :05
Jan-01-2002 12:00:05	0	Calculated, Good	5	Calculated, Good	Value1-No bound, Bad from :05 to :10
Jan-01-2002 12:00:10	5	Calculated, Good	5	Calculated, Good	
Jan-01-2002 12:00:15	5	Calculated, Good	5	Calculated, Good	

5.6.3.23.6 DurationGood data with uncertain data in the interval.

Start: Jan-01-2002 12:01:00 **End**: Jan-01-2002 12:01:30 **Interval**: 00:00:00

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:00	20*	Calculated, Good	25*	Calculated, Good	Value1-Uncertain from :10 to :20 Value1-Uncertain from :12 to :17

* Uncertain data should not be counted as good.

5.6.3.24 DurationBad

5.6.3.24.1 Description

The duration bad aggregate looks at the quality of a bounding value of the interval to determine what the quality is at the beginning of the interval. If no bounding value exists, the quality is assumed to be bad at the start of the interval. This aggregate only considers truly Bad values. Uncertain values are not considered bad for purposes of calculating this aggregate.

Whenever a raw value x with quality q is encountered from beginning to end within an interval, the quality is considered to be q until the next value, y, is encountered, at which point the quality becomes that of y, and so on.

The time is returned in seconds. No returned value will ever be uncertain or subnormal.

Each interval's aggregate is returned with timestamp of the start of the interval. *StatusCodes* are *Good, Calculated*.

Duration Bad is not simply the interval minus duration good, since the interval uncertain data.

5.6.3.24.2 DurationBad data with good bounding value.

Start: Jan-01-2002 12:00:10 **End**: Jan-01-2002 12:00:20 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:10	0	Calculated, Good	0	Calculated, Good	
Jan-01-2002 12:00:15	0	Calculated, Good	0	Calculated, Good	

5.6.3.24.3 DurationBad data with good bounding value with bad data in the interval.

Start: Jan-01-2002 12:00:35 **End**: Jan-01-2002 12:01:00 **Interval**: 00:00:05

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:00:35	0	Calculated, Good	0	Calculated, Good	
Jan-01-2002 12:00:40	5	Calculated, Good	3	Calculated, Good	
Jan-01-2002 12:00:45	5	Calculated, Good	3	Calculated, Good	
Jan-01-2002 12:00:50	0	Calculated, Good	0	Calculated, Good	
Jan-01-2002 12:00:55	0	Calculated, Good	0	Calculated, Good	

5.6.3.24.4 DurationBad data with uncertain data in the interval.

Start: Jan-01-2002 12:01:00 **End**: Jan-01-2002 12:01:30 **Interval**: 00:00:00

Timestamp	Historian 1		Historian 2		Notes
	Value	StatusCode	Value	StatusCode	
Jan-01-2002 12:01:00	0	Calculated, Good	0	Calculated, Good	

5.6.3.25 PercentGood

5.6.3.25.1 Description

This aggregate performs the following calculation:

percent_good = duration_good / interval_length * 100

Where:

duration_good is the result from the DURATIONGOOD aggregate, calculated using the interval supplied to PERCENTGOOD call.

Interval_length is the interval of the aggregates.

No returned value will ever be uncertain or subnormal.

Each interval's aggregate is returned with timestamp of the start of the interval. *StatusCodes* are *Good, Calculated*.

The interval_length is the entire sample interval, regardless of quality.

5.6.3.26 PercentBad

5.6.3.26.1 Description

This aggregate performs the following calculation:

percent_bad = duration_bad / interval_length * 100

Where:

duration_good is the result from the DURATIONBAD aggregate, calculated using the interval supplied to PERCENTBAD call.

Interval_length is the interval of the aggregates.

No returned value will ever be uncertain or subnormal.

Each interval's aggregate is returned with timestamp of the start of the interval. *StatusCodes* are *Good, Calculated*.

The interval_length is the entire sample interval, regardless of quality.

5.6.3.27 WorstQuality

5.6.3.27.1 Description

This aggregate returns the worst quality of the raw values in the interval. That is, *Bad* status are worse than *Uncertain*, which are worse than *Good*. No distinction is made between the specific reasons for the status.

This aggregate returns the worst *StatusCode* as the value of the aggregate.

The timestamp is always the start of the interval. The *StatusCodes* are *Good, Calculated*.

6 Client conventions

6.1 How clients may request timestamps

The OPC HDA COM based specifications allowed clients 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 must 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 this section to represent relative times in their user interfaces.

The time is considered to be a relative time local to the server. This means that all times are given in UTC time, computed from the current time on the server's local clock. The format for the relative time is:

```
keyword+/-offset+/-offset…
```

where keyword and offset are as specified in the table below. Whitespace is ignored. The time string must begin with a keyword. Each offset must be preceded by a signed integer that specifies the number and direction of the offset. If the integer preceding the offset is unsigned, 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, 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), it is backed up to the same time of day on February 28.

Note that the above does not hold for cases where one is adding or subtracting weeks or days, but only when 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.

Table 31 –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 32 –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.