AVEVA™
Historian Retrieval Guide
formerly Wonderware

Publication date: Thursday, December 3, 2020

Contact Information

AVEVA Group plc
High Cross
Madingley Road
Cambridge
CB3 0HB. UK

https://sw.aveva.com/

For information on how to contact sales and customer training, see https://sw.aveva.com/contact.

For information on how to contact technical support, see https://sw.aveva.com/support.

Contents

Welcome

This guide describes how to retrieve data that is stored by an AVEVA Historian server.

You can retrieve data by using:

- Transact-SQL queries.
- Historian Client tools for query construction, queries within Excel workbooks, and trend mapping.
- Historian Insight, a web-based tool for tag-based searches and charting. With Insight, you can save and reuse content (sets of tags and defined timeframes), and can share, embed, and download the results. Historian Insight is installed as a part of AVEVA Historian.
- Historian SDK.
- Tools that use the REST OData interface.

AVEVA Historian Documentation Set

The AVEVA Historian documentation set includes the following guides:

- *AVEVA System Platform Installation Guide*
  This guide provides information on installing the AVEVA Historian, including hardware and software requirements and migration instructions.
- *AVEVA Historian Concepts Guide*
  This guide provides an overview of the entire AVEVA Historian system and its key components.
- *AVEVA Historian Scenarios Guide*
  This guide discusses how to use AVEVA Historian to address some common customer scenarios.
- *AVEVA Historian Administration Guide*
  This guide describes how to administer and maintain an installed AVEVA Historian, such as configuring data acquisition and storage, managing security, and monitoring the system.
- *AVEVA Historian Retrieval Guide*
  This guide describes the retrieval modes and options that you can use to retrieve your data.
- *AVEVA Historian Database Reference*
  This guide provides documentation for all of the AVEVA Historian database entities, such as tables, views, and stored procedures.
- *AVEVA Historian Glossary*
  This guide provides definitions for terms used throughout the documentation set.

In addition, the *AVEVA License Manager Guide* describes the AVEVA License Manager and how to use it to install, maintain, and delete licenses and license servers on local and remote computers.

CHAPTER 1

About Data Retrieval

Through the Data Retrieval subsystem, AVEVA Historian receives SQL queries from clients, locates the requested data, performs necessary processing, and then returns the results.

For configuration and event data, retrieval is made possible by normal SQL queries, because these types of data are stored in SQL Server database tables. Historical data, however, must be retrieved from history blocks and then sent to clients as if it is stored in SQL Server tables.

To accomplish retrieval from both data repositories, the Data Retrieval subsystem includes:

- **An implementation of a SQL Server data provider.**
  This determines whether the requested data is saved in SQL Server tables or in history blocks.
- **Retrieval subsystem.**
  This subsystem is responsible for extracting the requested data from the history blocks and presenting to the AVEVA Historian OLE DB provider as "virtual" history tables.
- **A set of SQL Server extensions.**
  These are implemented as columns in the history tables. You can use these extensions to specify the nature of the rowset that is returned, such as the number of rows returned, the resolution of the data, or the retrieval mode.

For more information on data storage, see Managing Data Storage.

Data Retrieval Subsystem Features

Data Retrieval subsystem features include support for:

- **Queries with all tag types**
  You can include all tag types in the same query when retrieving from the History table. Any combination of tags can be submitted in a single query.
- **Both fixed- and variable-length strings**
- **FILETIME for time computations**
  All internal time computation and manipulation is done using the Win32 FILETIME type. The resolution of FILETIME is 100 nanoseconds.
- **Time handled as Universal Time Coordinated**
  All times are handled internally as UTC. Conversions to and from local time are handled going in and out of retrieval so the external interface is local time.
- **Retrieval of different versions**

Note: If you have an application that uses the older SQL Server datetime format, be aware that some rounding can occur as compared to the newer datetime2 format (for example, 3.3ms vs. 100ns).

History Blocks: A SQL Server Remote Data Source

History blocks are remote data sources used by AVEVA Historian. That is, they are data repositories that exist outside of a SQL Server database file (.MDF). Microsoft sometimes refers to these types of data sources as "non-local data stores."

All tag data is stored in history blocks. For more information on history blocks, see Managing Partitions and History Blocks in the *AVEVA Historian Administration Guide*.

OLE DB technology can be used to access data in any remote data store. This access is accomplished though a software component called an OLE DB provider.

Retrieval subsystem

The Retrieval subsystem does the following:

- Fetches history data from history blocks on disk.
- Formats data so that it can be passed up through the system to the AVEVA Historian OLE DB provider or other HCAL-enabled client applications.
- Returns information regarding the history blocks, such as the start and end dates and the location.

About the AVEVA Historian OLE DB Provider

OLE DB (short for "Object Linking and Embedding for Databases") is an application programming interface (API) that allows COM-based client applications to access data that is not physically stored in the SQL Server to which they are connecting.

OLE DB provides access to different types of data in a broader manner. By using OLE DB, you can simultaneously access data from a variety of sources. A query that accesses data from multiple, dissimilar data sources such as these is called a "heterogeneous" or "distributed" query.

SQL Server uses OLE DB to make linking data between the data sources easier. Through OLE DB, Microsoft SQL Server supports Transact-SQL queries against data stored in one or more SQL Server and heterogeneous databases without any need for specialized gateway server applications.

The interface required to access data in a non-local data store (such as the AVEVA Historian history blocks) is provided by a "virtual" server, called an OLE DB provider. OLE DB providers allow you to use the power of the SQL Server query processor to make linking data stored in the SQL Server databases and from history blocks much easier and more robust. Also, the AVEVA Historian OLE DB provider has a rich set of query capabilities.

The name of the AVEVA Historian OLE DB provider is "INSQL". The AVEVA Historian OLE DB provider is installed during AVEVA Historian installation and then associated, or linked, with the Microsoft SQL Server. For information on the syntax for linking the AVEVA Historian OLE DB provider, see *Linking the AVEVA Historian OLE DB Provider to the Microsoft SQL Server* on page 14.

Note: The INSQL OLE DB provider cannot be used in a standalone mode.

To access AVEVA Historian historical data using OLE DB, any COM-based client application must connect directly to the SQL Server and then specify to use the AVEVA Historian OLE DB provider in the syntax of the query.

When you execute a query and specify the AVEVA Historian OLE DB provider in the syntax, the Microsoft SQL Server parser will pass the appropriate parts of the data request to the AVEVA Historian OLE DB provider. The AVEVA Historian OLE DB provider will then interface with the retrieval service to locate the data store, extract the requested information, and return the data to the Microsoft SQL Server as a rowset. Microsoft SQL Server will perform any other processing required on the data and return the data to the client application as a result set and a set of output parameters, if applicable.

The AVEVA Historian OLE DB provider must be present on the server running Microsoft SQL Server. The set of Transact-SQL operations that can be used to retrieve data in the history blocks depends on the capabilities of the AVEVA Historian OLE DB provider.

For more information on OLE DB, see your Microsoft documentation.

Extension Tables for History Data

Many of Historian's tables are implemented as extension tables. That is, they are logical tables that are actually populated from data in history blocks.

Note: Extension tables are also called remote tables.

Data access from the history blocks is made possible by SQL Server's OLE DB provider technology. Client applications must connect directly to the Microsoft SQL Server and then specify to use the AVEVA Historian OLE DB provider in the syntax of the query.

The extension tables are:

History	[INSQL].Runtime.dbo.History
Live	[INSQL].Runtime.dbo.Live
AnalogSummaryHistory	[INSQL].Runtime.dbo.AnalogSummaryHistory
StateSummaryHistory	[INSQL].Runtime.dbo.StateSummaryHistory
HistoryBlock	[INSQL].Runtime.dbo.HistoryBlock
Events	[INSQL].Runtime.dbo.Events

For more information on the history extension tables, see History Tables and Views in the *Historian Database Reference*.

Legacy Process Data Extension Tables

These are legacy (backward compatible) extension tables for process data:

AnalogHistory	[INSQL].Runtime.dbo.AnalogHistory
DiscreteHistory	[INSQL].Runtime.dbo.DiscreteHistory
StringHistory	[INSQL].Runtime.dbo.StringHistory
AnalogLive	[INSQL].Runtime.dbo.AnalogLive
DiscreteLive	[INSQL].Runtime.dbo.DiscreteLive
StringLive	[INSQL].Runtime.dbo.StringLive

The AnalogHistory, DiscreteHistory, StringHistory, and History tables are the only tables which are updateable. The remaining tables are read-only.

For more information about these tables, see Backward Compatibility Entities in the *Historian Database Reference*.

Legacy Event Data Extension Tables

These are legacy (backward compatible) extension tables for events:

v_AlarmEventHistoryInternal2	[INSQL].Runtime.dbo.LegacyAlarmEventHistory
v_AlarmHistory	[INSQL].Runtime.dbo.LegacyAlarmHistory
v_AlarmHistory2	[INSQL].Runtime.dbo.LegacyAlarmHistory2
v_EventHistory	[INSQL].Runtime.dbo.LegacyEventHistory
v_AlarmEventHistory2	Same as v_AlarmEventHistoryInternal2

Linking the AVEVA Historian OLE DB Provider to the Microsoft SQL Server

Because the AVEVA Historian OLE DB provider retrieves data from the history blocks and presents it to Microsoft SQL Server as a table, it can be thought of as a type of server. The AVEVA Historian OLE DB provider must be added to the Microsoft SQL Server as a "linked" server before it can be used to process queries.

This linking is performed automatically during the AVEVA Historian installation. If, for some reason, you need to re-link the AVEVA Historian OLE DB provider to the Microsoft SQL Server, the statements for linking are as follows:

```
sp_addlinkedserver
    @server = 'INSQL',
    @srvproduct = '',
    @provider = 'INSQL'
    go
    sp_serveroption 'INSQL','collation compatible',true
    go
    sp_addlinkedsrvlogin 'INSQL','TRUE',NULL,NULL,NULL
    go
```

"INSQL" is the name of the AVEVA Historian OLE DB provider as the linked server. Use this name to specify the AVEVA Historian OLE DB provider in a query.

To perform joins between the legacy analog history tables and discrete history tables, the installation program also creates an alias for the same AVEVA Historian OLE DB provider:

```
sp_addlinkedserver
    @server = 'INSQLD',
    @srvproduct = '',
    @provider = 'INSQL'
    go
    sp_serveroption 'INSQLD','collation compatible',true
    go
    sp_addlinkedsrvlogin 'INSQLD','TRUE',NULL,NULL,NULL
    go
```

For example, if you want to execute a query that performs this type of join, use the normal alias in specifying the first table (the analog history table), and use the second alias in specifying the second table (the discrete history table, hence the "D" added to the alias name).

AVEVA Historian I/O Server

The AVEVA Historian I/O Server (aahIOSvrSvc.exe) is the interface for clients to access current data using the SuiteLink protocol. The AVEVA Historian I/O Server can update items with current values for given topics, providing "real-time" I/O Server functionality.

The AVEVA Historian I/O Server is pre-configured with a single topic, Tagname. The AVEVA Historian I/O Server will listen for clients (such as WWClient or WindowViewer™) that are attempting to establish a connection using the pre-configured topic. After a client connects with the AVEVA Historian I/O Server, a "hot" link is established between the client and the AVEVA Historian I/O Server. For more information on I/O Server addressing conventions, see I/O Server Addressing in the *AVEVA Historian Administration Guide.*

For example, the AVEVA Historian I/O Server could be used by InTouch WindowViewer to access system tag values provided by the AVEVA Historian to monitor system health. You could configure WindowViewer to generate an alarm when abnormal behavior is detected within the AVEVA Historian.

By default, the AVEVA Historian I/O Server runs as a Windows service and can be started and stopped using the System Management Console. You can also monitor the AVEVA Historian I/O Server from within the System Management Console. For more information on the System Management Console, see About Administrative Tools in the *AVEVA Historian Administration Guide.*

The AVEVA Historian I/O Server is a read-only server; clients cannot update data.

The AVEVA Historian I/O Server sends the original OPC quality as it was stored in the AVEVA Historian. The OPC quality remains the same throughout the system, including storage, retrieval, and the AVEVA Historian I/O Server.

Using SELECT to Retrieve Data

The most common AVEVA Historian query is a SELECT statement:

```
SELECT select_list
    FROM table_source
    WHERE search_condition
        [ GROUP BY group_by_expression ]
        [ HAVING search_condition ]
        [ ORDER BY order_expression [ ASC | DESC ] ]
```

A WHERE clause is mandatory when issuing a SELECT query against any extension table except HistoryBlock.

There are four variations for issuing a SELECT statement to the AVEVA Historian OLE DB provider to retrieve history data:

- *Using the Four-Part Naming Convention* on page 15
- *Using an AVEVA Historian OLE DB Provider View* on page 16
- *Using the OPENQUERY Function* on page 17
- *Using the OPENROWSET Function* on page 17

You should use the four-part name or a provider view to specify the extension table, whenever possible. However, there are instances when the OPENQUERY or OPENROWSET function must be used, such as for queries on wide tables.

For general information on creating SQL queries, see your Microsoft SQL Server documentation.

Using the Four-Part Naming Convention

The linked server name is simply a name by which the AVEVA Historian OLE DB provider is known to the Microsoft SQL Server. In order for a query to be passed on to the AVEVA Historian OLE DB provider, you must specify the linked server name and the extension table name as part of a four-part naming convention.

For example, this query specifies to retrieve data from the History extension table in the AVEVA Historian OLE DB provider:

```
SELECT * FROM INSQL.Runtime.dbo.History
    WHERE TagName = 'SysTimeSec'
        AND DateTime >= '2001-09-12 12:59:00'
        AND DateTime <= '2001-09-12 13:00:00'
```

The four-part naming convention is described in the following table:

Part Name	Description
linked_server	Linked server name. By default, INSQL.
catalog	Catalog in the OLE DB data source that contains the object from which you want to retrieve data. For Microsoft SQL Server type databases, this is the name of the database. To use the AVEVA Historian OLE DB provider, the catalog name will always be "Runtime."
schema	Schema in the catalog that contains the object. For Microsoft SQL Server type databases, this is the name of the login ID for accessing the data. To use the AVEVA Historian OLE DB provider, the catalog name will always be "dbo."
object_name	Data object that the OLE DB provider can expose as a rowset. For the AVEVA Historian OLE DB provider, the object name is the name of the remote table that contains the data you want to retrieve. For example, the History table.

In the case of four-part queries, SQL Server produces the statement that is sent to the AVEVA Historian OLE DB provider from the statement that the user executes. Sometimes this produced statement is incorrect, too complex, or lacks portions of the WHERE clause required for the AVEVA Historian OLE DB provider to return data.

A typical error message when executing unsupported syntax is:

```
Server: Msg 7320, Level 16, State 2, Line 1
Could not execute query against OLE DB provider 'INSQL'.
[OLE/DB provider returned message: InSQL did not receive a WHERE clause from SQL
Server. If one was specified, refer to the InSQL OLE DB documentation]
```

For four-part queries against non-English SQL Servers running on non-English operating systems, the default date format might differ from the English versions. For example, for a French or German SQL Server running on the corresponding operating system, the date/time in a four-part query must be:

yyyy-dd-mm hh:mm:ss.fff

For example:

2003-28-09 09:00:00.000

The default SQL date format is dependent on SQL Server and not on the operating system used. However, you can modify the format using the SQL Server Convert() method. The output of this method can be determined by the regional settings configured for the operating system.

Using an AVEVA Historian OLE DB Provider View

Microsoft SQL Server views have been provided that will access each of the extension tables, eliminating the need to type the four-part server name in the query. These views are named the same as the provider table name.

Note: Backward compatibility views are named according to the v_ProviderTableName convention.

For example:

```
SELECT * FROM History
   WHERE TagName = 'SysTimeSec'
      AND DateTime >= '2001-09-12 12:59:00'
      AND DateTime <= '2001-09-12 13:00:00'
```

Using the OPENQUERY Function

You can use the linked server name in an OPENQUERY function to retrieve data from an extension table. The OPENQUERY function is required for retrieving from the wide table. For example:

```
SELECT * FROM OPENQUERY(INSQL, 'SELECT * FROM History
   WHERE TagName = "SysTimeSec"
      AND DateTime >= "2001-09-12 12:59:00"
      AND DateTime <= "2001-09-12 13:00:00"
      ')
```

The following example retrieves data from a wide table:

```
SELECT * FROM OPENQUERY(INSQL, 'SELECT DateTime, SysTimeSec
   FROM WideHistory
      WHERE DateTime >= "2001-09-12 12:59:00"
         AND DateTime <= "2001-09-12 13:00:00"
      ')
```

The OPENQUERY portion of the statement is treated as a table by SQL Server, and can also be used in joins, views, and stored procedures. SQL Server sends the quoted statement, unchanged and as a string, to the AVEVA Historian OLE DB provider. Consequently, only the syntax that the AVEVA Historian OLE DB provider can parse is supported. Also, be sure that you do not exceed the 8000 character limit for the statement. Consider the following example:

```
SELECT * FROM OpenQuery(INSQL, 'XYZ')
```

where "XYZ" is the statement to pass. You should be sure that the value of "XYZ" is not more than 8000 characters. This limit is most likely to cause a problem if you are querying many tags from a wide table.

Also, you should supply the datetime in an OPENQUERY statement in the following format:

yyyy-mm-dd hh:mm:ss.fff

For example:

2001-01-01 09:00:00.000

You cannot use variables in an OPENQUERY statement. For more information, see -old-Using Variables with the Wide Table.

Using the OPENROWSET Function

The linked server name can be used as an input parameter to an OPENROWSET function. The OPENROWSET function sends the OLE DB provider a command to execute. The returned rowset can then be used as a table or view reference in a Transact-SQL statement. For example:

```
SELECT * FROM OPENROWSET('INSQL',' ', 'SELECT DateTime, Quality, QualityDetail,
Value
   FROM History
   WHERE TagName in ("SysTimeSec")
   AND DateTime >= "2001-09-12 12:59:00"
   AND DateTime <= "2001-09-12 13:00:00"
   ')
```

Note: If the OpenRowSet/OpenDatasource component is turned off as part of the security configuration for the server, you will receive an error when you try to run this query. If necessary, a system administrator can reset SQL Server settings to enable use of ad hoc queries by executing the sp_configure command.

Supported Syntax Options

The following table indicates the syntax options that are available for queries that use either the four-part naming convention (or corresponding view name) or the OPENQUERY function.

Syntax Element	Four-Part Query	OPENQUERY
ORDER BY	Yes	No. Does not work within the OPENQUERY function. However, will work if used outside of the function.
GROUP BY	Yes	No
TagName IN (..)	Yes	Yes
TagName LIKE '..'	Yes	Yes
Date and time functions (for example, DateAdd)	Yes	Yes
MIN, MAX, AVG, SUM, STDEV	Yes	MIN, MAX, AVG, SUM only
Sub-SELECT with one normal SQL Server table and one extension table	Yes, with restrictions	No
Sub-SELECT with two extension tables	No	No

Unsupported or Limited Syntax Options

The AVEVA Historian OLE DB provider does not support certain syntax options in queries. In general, these limitations are due to underlying limitations in the current Microsoft SQL Server OLE DB Provider implementation.

For general information on creating SQL queries, see your Microsoft SQL Server documentation.

No Notion of Client Context

The OLE DB provider has no notion of a client context. The OLE DB provider is entirely stateless, and there is no persistence across queries in the same connection. This means that you must set the value of a AVEVA Historian time domain extension (for example, cycle count) each time you execute a query.

Also, the OLE DB provider cannot continuously return data (similar to a "hot" link in InTouch HMI software). The OLE DB specification (as defined by Microsoft) does not permit a provider to return rows to a consumer without a request from the consumer.

Limitations on Wide Tables

Wide tables do not have a fixed schema, but a schema which varies from query to query. They are transient tables, existing for the duration of one query only. For this reason, they must be accessed using the OPENQUERY function, which bypasses many of the tests and requirements associated with fixed tables. Wide tables support up to 1024 columns.

For more information on wide tables, see "Wide" History Table Format in the *AVEVA Historian Database Reference*.

LIKE Clause Limitations

The LIKE clause is only supported for the TagName and Value columns. The syntax " `... Value LIKE 'a string' ...` " is only supported for a string table. For example:

```
SELECT TagName, Value FROM History
   WHERE TagName LIKE 'Sys%'
      AND DateTime > '1999-05-24 14:30:00'
      AND DateTime < '1999-05-24 14:32:00'
```

IN Clause Limitations

If you are querying analog, discrete, or string tags from the AnalogTag, DiscreteTag, or StringTag tables (respectively), you cannot use the LIKE clause within an IN clause to condition the tagname unless you are returning the vValue column. This restriction applies if you are using the four-part naming convention or an extension table view.

For example:

```
SELECT DateTime, TagName, vValue, Quality, QualityDetail
   FROM History
      WHERE TagName IN (SELECT TagName FROM StringTag WHERE TagName LIKE
      'SysString')
         AND DateTime >='2001-06-21 16:00:00.000'
         AND DateTime <='2001-06-21 16:40:00.000'
         AND wwRetrievalMode = 'Delta'
```

However, it is more efficient to use an INNER REMOTE JOIN to achieve the same results. For more information, see Using an INNER REMOTE JOIN.

OR Clause Limitations

You cannot use the OR clause to specify more than one condition for a time domain extension. For more information, see *AVEVA Historian Time Domain Extensions* on page 24.

Using Joins within an OPENQUERY Function

Joins are not supported within a single OPENQUERY statement. For example, the following query contains an implicit join between the Tag and Live tables, and will fail:

```
SELECT * FROM OPENQUERY(INSQL, 'SELECT v.DateTime, v.TagName, v.Value,
t.Description
   FROM Tag t, Live v
      WHERE t.TagName LIKE "%Date%"
```

```
        AND v.TagName = t.TagName
    ')
```

A workaround is to place the join outside of the OPENQUERY. For example:

```
SELECT v.DateTime, v.TagName, v.Value, t.Description
    FROM OPENQUERY(INSQL, 'SELECT DateTime, TagName, Value
    FROM Live
        WHERE TagName LIKE "%Date%"
    ') v, Tag t
        WHERE v.TagName = t.TagName
```

Explicit joins are also not supported within OPENQUERY. For example, the following query will fail:

```
SELECT * FROM OPENQUERY(INSQL, 'SELECT v.DateTime, v.TagName, v.Value, e.Unit
    FROM Live v
    JOIN AnalogTag t ON v.TagName = t.TagName
    JOIN EngineeringUnit e ON t.EUKey = e.EUKey
        WHERE v.TagName LIKE "%Date%"
    ')
```

A work-around is to place the join outside the OPENQUERY. For example:

```
SELECT v.DateTime, v.TagName, v.Value, e.Unit
    FROM OPENQUERY(INSQL, 'SELECT DateTime, TagName, Value FROM Live
        WHERE TagName LIKE "%Date%"
    ') v
    JOIN AnalogTag t ON v.TagName = t.TagName
    JOIN EngineeringUnit e ON t.EUKey = e.EUKey
        ORDER BY t.TagName
```

In general, use four-part syntax wherever possible. All of the previous queries are more conveniently expressed in four-part syntax. For example, the syntax for the preceding query would be:

```
SELECT v.DateTime, v.TagName, v.Value, e.Unit
    FROM INSQL.Runtime.dbo.History v
    JOIN AnalogTag t ON v.TagName = t.TagName
    JOIN EngineeringUnit e ON t.EUKey = e.EUKey
        WHERE v.TagName LIKE '%Date%'
        ORDER BY t.TagName
```

Using Complicated Joins

You can only use simple joins between SQL Server tables and the AVEVA Historian OLE DB extension tables. Joins typically require use of the INNER REMOTE JOIN syntax.

For an example of the INNER REMOTE JOIN syntax, see -old-Using an INNER REMOTE JOIN.

Using a Sub-SELECT with a SQL Server Table and an Extension Table

Using a sub-SELECT with a query on a normal SQL Server table and an extension table should be avoided; it is very inefficient due to the way SQL Server executes the query. For example:

```
SELECT TagName, DateTime, Value
    FROM INSQL.Runtime.dbo.History
        WHERE TagName IN (select TagName FROM SnapshotTag WHERE EventTagName =
        'SysStatusEvent')
            AND DateTime = '2001-12-20 0:00'
```

Instead, it is recommended that you use the INNER REMOTE JOIN syntax:

```
SELECT h.TagName, DateTime, Value
   FROM SnapshotTag st INNER REMOTE JOIN INSQL.Runtime.dbo.History h
      ON st.TagName = h.TagName
         AND EventTagName = 'SysStatusEvent'
         AND DateTime = '2001-12-20 0:00'
```

The results are:

TagName	DateTime	Value
SysPerfCPUTotal	2001-12-20 00:00:00.000	15.0
SysSpaceMain	2001-12-20 00:00:00.000	1302.0

In general, use the following pattern for INNER REMOTE JOIN queries against the historian is:

<SQLServerTable> INNER REMOTE JOIN <HistorianExtensionTable>

For more information on INNER REMOTE JOIN, see your Microsoft SQL Server documentation.

WHERE Clause Anomalies

In some rare cases, the SQL Server query processor truncates the WHERE clause in an attempt to optimize the query. If you execute a query with a WHERE clause, but an error message is returned stating that no WHERE clause was received by the SQL Server, simply add another condition clause to the query.

For example, in the following query, the SQL Server query processor optimizes out the WHERE clause, because it is superfluous.

```
SELECT DateTime, Value, QualityDetail
   FROM History
      WHERE TagName LIKE '%'
```

A workaround is to add another condition clause. For example:

```
SELECT DateTime, Value, QualityDetail
   FROM History
      WHERE TagName LIKE '%'
      AND wwRetrievalMode = 'delta'
```

CONVERT Function Limitations

The CONVERT function is not supported on the Value column in an OPENQUERY statement. If you are using OPENQUERY on the History table, you must filter on the Value column outside of the query.

In the following example, the value of the Value column is converted to a float. Note that no string tags are included in the query.

```
SELECT * FROM OpenQuery(INSQL, 'SELECT DateTime, Quality, OPCQuality,
QualityDetail, Value, vValue, TagName
   FROM History
      WHERE TagName IN ("SysTimeMin", "SysPulse")
         AND DateTime >= "2001-12-30 04:00:00.000"
         AND DateTime <= "2001-12-30 09:00:00.000"
```

```
        AND wwRetrievalMode = "Delta"
    ')
    WHERE convert(float, vValue) = 20.0
```

You can also use the following formats:

```
WHERE convert(float, vValue) = 0
WHERE convert(float, vValue) = 0.0
WHERE convert(float, vValue) = 1.0
WHERE convert(float, vValue) = 1
WHERE convert(float, vValue) = 20
WHERE convert(float, vValue) = 2.0000e01
```

The following example includes a string tag and converts the vValue value to a char or varchar datatype. All values returned can be converted to a string.

```
SELECT * FROM OpenQuery(INSQL, 'SELECT DateTime, Quality, OPCQuality,
QualityDetail, Value, vValue, TagName
    FROM History
        WHERE TagName IN ("SysString", "SysTimeMin", "SysPulse")
            AND DateTime >= "2001-12-30 04:00:00.000"
            AND DateTime <= "2001-12-30 09:00:00.000"
            AND wwRetrievalMode = "Cyclic"
            AND wwCycleCount = 300
    ')
    WHERE convert(varchar(30), vValue) = '2001-12-30 14:00:00'
```

You can also use the following formats:

```
WHERE convert(varchar(30), vValue) = '20'
WHERE convert(varchar(30), vValue) = '1'
WHERE convert(varchar(30), vValue) = '0'
```

SQL Server Optimization of Complex Queries

The SQL Server query optimizer may incorrectly parse a complex query and not send certain query criteria to the Historian OLE DB provider for handling. This can cause unexpected results for the data.

If you suspect that this is happening, use SQL Server Management Studio tools to examine the query plan that the optimizer is using and then modify your query so that the needed criteria gets directed to the Historian OLE DB provider.

For example, the following query will be incorrectly parsed:

```
SELECT GETDATE()
DECLARE @TagList TABLE (TagName nvarchar(256))
INSERT @TagList
    SELECT 'SysTimeSec' UNION
    SELECT 'SysPerfCPUTotal'
-- Prevent the TagName criteria from being sent to the Historian OLE DB provider
(incorrect)
SELECT DateTime, h.vValue, h.TagName
    FROM History h
    INNER REMOTE JOIN @TagList l
    ON h.TagName = l.TagName
    WHERE DateTime >= DATEADD(hour,-1,GETDATE())
        AND DateTime < GETDATE()
        AND wwRetrievalMode = 'AVG'
        AND wwCycleCount=1
```

```
GO
```

To correct this issue, rewrite the query so that the tagname criteria is passed to the Historian OLE DB provider correctly.

```
SELECT GETDATE()
DECLARE @TagList TABLE (TagName nvarchar(256))
INSERT @TagList
    SELECT 'SysTimeSec' UNION
    SELECT 'SysPerfCPUTotal'
-- Force the TagName criteria to be sent to the InSQL OLE DB Provider (correct)
SELECT DateTime, h.vValue, h.TagName
    FROM @TagList l
    INNER REMOTE JOIN History h
    ON h.TagName = l.TagName
    WHERE DateTime >= DATEADD(hour,-1,GETDATE())
        AND DateTime < GETDATE()
        AND wwRetrievalMode = 'AVG'
        AND wwCycleCount=1
GO
```

Using Columns of a Variant Type with Functions

If you use a column of a variant type as the parameter for some functions, SQL Server returns a syntax error. However, the error is not passed to the Historian OLE DB provider to return to clients.

For example, in the following query, the rounding is specified for the vValue column, which is of type variant. The query does not work, but no error is returned by the Historian OLE DB provider.

```
SELECT DateTime, round(vValue, 2)
    FROM History
    WHERE TagName IN ('SysTimeSec')
        AND DateTime = getdate()
        AND wwRetrievalMode = 'Cyclic'
```

Using StartDateTime in the Query Criteria

You cannot use StartDateTime in the query criteria instead of DateTime. For example, the following query works, except that it does not apply the StartDateTime >= @StartDate clause.

```
SET NOCOUNT ON
DECLARE @StartDate DateTime
DECLARE @EndDate DateTime
SET @StartDate = DateAdd(mi,-30,GetDate())
SET @EndDate = GetDate()
SET NOCOUNT OFF
SELECT History.TagName, DateTime = convert(nvarchar, DateTime, 21), Value,
vValue, StateTime, StartDateTime
    FROM History
        WHERE History.TagName IN ('Reactor1Level')
            AND wwRetrievalMode = 'RoundTrip'
            AND wwStateCalc = 'AvgContained'
            AND vValue = convert(SQL_VARIANT, '1')
            AND wwCycleCount = 1
            AND wwTimeStampRule = 'Start'
            AND wwQualityRule = 'Good'
```

```
AND wwFilter = 'ToDiscrete(5.0,>)'
AND wwVersion = 'Latest'
AND DateTime >= @StartDate
AND DateTime <= @EndDate
AND StartDateTime >= @StartDate
```

Comparison Statements and NULL Values

SQL Server returns an error for a query that contains a comparison statement like 'Value > 0' whenever a NULL is returned. Be sure that you always include 'AND Value IS NOT NULL', so that the NULL values are filtered out.

OPENQUERY and Microsoft Query

Microsoft Query is not able to process an OPENQUERY statement.

AVEVA Historian Time Domain Extensions

Data in the extension tables can be manipulated by using normal Transact-SQL code, as well as the specialized SQL time domain extensions provided by the AVEVA Historian. The AVEVA Historian extensions provide an easy way to query time-based data from the history tables. They also provide additional functionality not supported by Transact-SQL.

The time domain extensions are:

- wwCycleCount
- wwEdgeDetection
- wwFilter
- wwInterpolationType
- wwOption
- wwQualityRule
- wwResolution
- wwRetrievalMode
- wwStateCalc
- wwTimeDeadband
- wwTimeZone
- wwValueDeadband
- wwVersion
- wwTimeStampRule
- wwValueSelector

Note: The wwParameters and wwMaxStates parameters are reserved for future use. The wwRowCount parameter is still supported, but is deprecated in favor of wwCycleCount.

The extensions are implemented as "virtual" columns in the extension tables. When you query an extension table, you can specify values for these column parameters to manipulate the data that will be returned. You will need to specify any real-time extension parameters each time that you execute the query.

For example, you could specify a value for the wwResolution column in the query so that a resolution is applied to the returned data set:

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-02 10:00:00'
         AND DateTime <= '2001-12-02 10:02:00'
         AND Value >= 50
         AND wwResolution = 10
```

```
AND wwRetrievalMode = 'cyclic'
```

Because the extension tables provide additional functionality that is not possible in a normal SQL Server, certain limitations apply to the Transact-SQL supported by these tables. For more information, see *Unsupported or Limited Syntax Options* on page 18.

Although the Microsoft SQL Server may be configured to be case-sensitive, the values for the virtual columns in the extension tables are always case-insensitive.

Note: You cannot use the IN clause or OR clause to specify more than one condition for a time domain extension. For example, "`wwVersion IN ('original', 'latest')`" and "`wwRetrievalMode = 'Delta' OR wwVersion = 'latest'`" are not supported.

For general information on creating SQL queries, see your Microsoft SQL Server documentation.

CHAPTER 2

Data Retrieval Options

You can use a variety of retrieval modes and options to suit different reporting needs and applications.

Understanding Retrieval Modes

Different retrieval modes allow you to access the data stored in an AVEVA Historian in different ways. For example, if you retrieve data for a long time period, you might want to retrieve only a few hundred evenly spaced data points to minimize response time. For a shorter time period, you might want to retrieve all values that are stored on the server to get more accurate results.

An AVEVA Historian with a version earlier than 9.0 supports two retrieval modes:

- *Cyclic Retrieval* on page 27
- *Delta Retrieval* on page 30

An AVEVA Historian with a version of 9.0 or higher supports various additional modes:

- *Full Retrieval* on page 35
- *Interpolated Retrieval* on page 37
- *"Best Fit" Retrieval* (see "*Best Fit Retrieval*" on page 41)
- *Average Retrieval* on page 46
- *Minimum Retrieval* on page 50
- *Maximum Retrieval* on page 55
- *Integral Retrieval* on page 59
- *Slope Retrieval* on page 62
- *Counter Retrieval* on page 65
- *ValueState Retrieval* on page 70

An AVEVA Historian with a version of 10.0 or higher supports the following additional mode:

- *RoundTrip Retrieval* on page 75

An AVEVA Historian version 11.6.14101 or higher supports the following additional mode:

- *Predictive Retrieval* (see "*Predictive Filter*" on page 86)

An AVEVA Historian with a version of 17.3.100 or higher supports the following additional mode:

- Bounding Value Retrieval

Cyclic Retrieval

Cyclic retrieval is the retrieval of stored data for the given time period based on a specified cyclic retrieval resolution, regardless of whether or not the value of the tag(s) has changed. It works with all types of tags. Cyclic retrieval produces a virtual rowset, which may or may not correspond to the actual data rows stored on the AVEVA Historian.

In cyclic retrieval, one row is returned for each "cycle boundary." You specify the number of cycles either directly or by means of a time resolution, that is, the spacing of cycle boundaries in time. If you specify a number of cycles, the AVEVA Historian returns that number of rows, evenly spaced in time over the requested period. The cyclic resolution is calculated by dividing the requested time period by the number of cycle boundaries. If you specify a resolution, the number of cycles is calculated by dividing the time period by the resolution.

If no data value is actually stored at a cycle boundary, the last value before the boundary is returned.

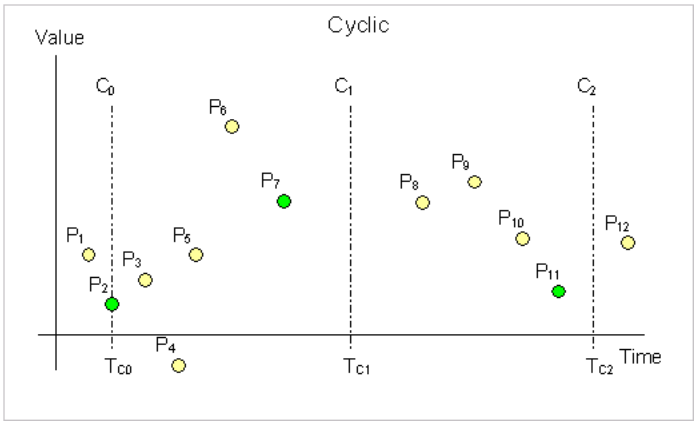
Beginning with AVEVA System Platform 2014 R2 SP1, Historian cyclic storage rules improve the handling of "slow rate change" data tags. Instead of delaying posts to the database if tag values do not arrive in a timely manner, new rules define a cyclic timeout when the database will be updated anyway. That timeout is typically one-half the period for the tag's cycle storage rate or the database server's maximum cyclic storage timeout, whichever is shorter. The time out is controlled by a the system parameter MaxCyclicStorageTimeout. For more information, see System Parameters in the *AVEVA Historian Administration Guide*.

The default retrieval mode is cyclic for retrieval from analog tables, including analog and state summary tables.

Cyclic retrieval is fast and therefore consumes little server resources. However, it may not correctly reflect the stored data because important process values (gaps, spikes, etc.) might fall between cycle boundaries. For an alternative, see *Best Fit Retrieval* (see "*Best Fit Retrieval*" on page 41).

Cyclic Retrieval - How It Works

The following illustration shows how values are returned for cyclic retrieval:



Data is retrieved in cyclic mode with a start time of T_{C0} and an end time of T_{C2}. The resolution has been set in such a way that the historian returns data for three cycle boundaries at T_{C0}, T_{C1}, and T_{C2}. Each dot in the graphic represents an actual data point stored on the historian. From these points, the following are returned:

- At T_{C0}: P_2, because it falls right on the cycle boundary
- At T_{C1}: P_7, because it is the last point before the cycle boundary
- At T_{C2}: P_{11}, for the same reason

Cyclic Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in cyclic retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *TimeStamp Rule (wwTimeStampRule)* on page 107, for AVEVA Historian 9.0 and above
- *Quality Rule (wwQualityRule)* on page 111

Cyclic Retrieval - Query Example

To use the cyclic retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'Cyclic'
```

For example, the following query returns data values for the analog tag 'ReactLevel'. If you do not specify a wwCycleCount or wwResolution, the query will return 100 rows (the default).

```
SELECT DateTime, Sec = DATEPART(ss, DateTime), TagName, Value
   FROM History
      WHERE TagName = 'ReactLevel'
         AND DateTime >= '2001-03-13 1:15:00pm'
         AND DateTime <= '2001-03-13 2:15:00pm'
         AND wwRetrievalMode = 'Cyclic'
```

The results are:

DateTime	Sec	TagName	Value
2001-03-13 13:15:00.000	0	ReactLevel	1775.0
2001-03-13 13:15:00.000	36	ReactLevel	1260.0
2001-03-13 13:16:00.000	12	ReactLevel	1650.0
2001-03-13 13:16:00.000	49	ReactLevel	1280.0
2001-03-13 13:17:00.000	25	ReactLevel	1525.0
2001-03-13 13:18:00.000	1	ReactLevel	585.0
2001-03-13 13:18:00.000	38	ReactLevel	1400.0
2001-03-13 13:19:00.000	14	ReactLevel	650.0
2001-03-13 13:19:00.000	50	ReactLevel	2025.0
2001-03-13 13:20:00.000	27	ReactLevel	765.0
2001-03-13 13:21:00.000	3	ReactLevel	2000.0
2001-03-13 13:21:00.000	39	ReactLevel	830.0
2001-03-13 13:22:00.000	16	ReactLevel	1925.0
...			
(100 row(s) affected)			

Cyclic Retrieval - Initial Values

No special handling is done for initial values. The initial value will behave like a normal cycle boundary at the start time. For information on initial values, see *Delta Retrieval - Initial Values* on page 34.

Cyclic Retrieval - Handling NULL Values

No special handling is done for NULL values. They are returned just like any other value.

Delta Retrieval

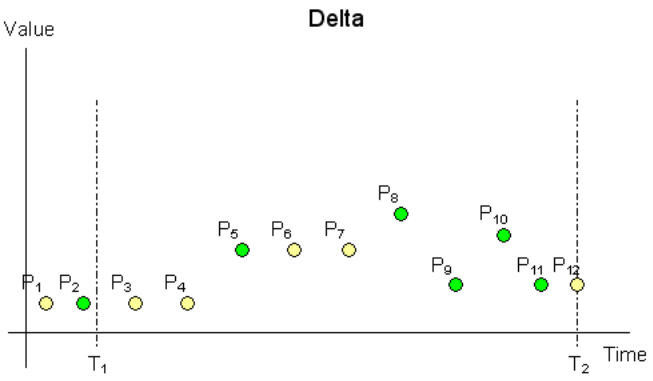
Delta retrieval, or retrieval based on exception, is the retrieval of only the changed values for a tag(s) for the given time interval. That is, duplicate values are not returned. It works with all types of tags.

Delta retrieval always produces a rowset comprised of only rows that are actually stored on the historian; that is, a delta query returns all of the physical rows in history for the specified tags, over the specified period, minus any duplicate values. If there is no actual data point at the start time, the last data point before the start time is returned.

Delta retrieval is the default mode for discrete and string tables and from the History table.

Delta Retrieval - How It Works

The following illustration shows how values are returned for delta retrieval:



Data is retrieved in delta mode with a start time of T_1 and an end time of T_2. Each dot in the graphic represents an actual data point stored on the historian. From these points, the following are returned:

- P_2, because there is no actual data point at T_1
- P_5, P_8, P_9, P_{10}, and P_{11}, because they represent changed values during the time period

For delta retrieval for replicated summary tags on a tier-2 historian, if a point with doubtful quality is returned as the result of a value selection from an input summary point with a contained gap, the same point can be returned again with good quality if the same value is selected again from the next input summary point that has good quality.

Delta Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in delta retrieval mode. For more information, see the following sections:

- *Time Deadband (wwTimeDeadband)* on page 97

- *Value Deadband (wwValueDeadband)* on page 101
- *History Version (wwVersion)* on page 104
- *Quality Rule (wwQualityRule)* on page 111

Delta Retrieval - Query Examples

To use the delta retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'Delta'
```

For examples, see the following:

- *Delta Retrieval - Query 1* on page 31
- *Delta Retrieval - Query 2* on page 32
- *Delta Retrieval - Query 3* on page 33
- *Delta Retrieval - Query 4* on page 33

Delta Retrieval - Query 1

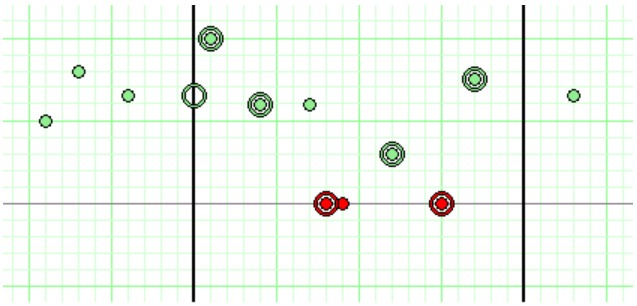
As an example of how delta mode works, consider the following query:

```
SELECT TagName, DateTime, Value, QualityDetail
   FROM History
      WHERE TagName = 'A001'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwRetrievalMode = 'Delta'
```

This query can be run against the following sample data:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:11	1.0	192
A001	2009-09-12 00:13	1.6	192
A001	2009-09-12 00:16	1.3	192
A001	2009-09-12 00:21	2.0	192
A001	2009-09-12 00:24	1.2	192
A001	2009-09-12 00:27	1.2	192
A001	2009-09-12 00:28	0.0	249
A001	2009-09-12 00:29	0.0	249
A001	2009-09-12 00:32	0.6	192
A001	2009-09-12 00:35	0.0	249
A001	2009-09-12 00:37	1.5	192
A001	2009-09-12 00:43	1.3	192

A graphical representation of the data is as follows:



The results are:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:20	1.3	192
A001	2009-09-12 00:21	2.0	192
A001	2009-09-12 00:24	1.2	192
A001	2009-09-12 00:28	NULL	249
A001	2009-09-12 00:32	0.6	192
A001	2009-09-12 00:35	NULL	249
A001	2009-09-12 00:37	1.5	192

The sample data points and the results are mapped on the following chart. Only the data falling between the time start and end marks at 2009-09-12 00:20 and 2009-09-12 00:40 (shown on the chart as dark vertical lines) are returned by the query.

Because there is no value that matches the start time, an initial value at 2009-09-12 00:20 is returned in the results based on the value of the preceding data point at 2009-09-12 00:16. Because there is no change in the value at 2009-09-12 00:27 from the value at 2009-09-12 00:24, the data point appears on the chart but does not appear in the results. Similarly, the second 0.0 value at 2009-09-12 00:29 is also excluded from the results.

You can further control the number of rows returned by using the wwTimeDeadband, wwValueDeadband, and wwCycleCount extensions. The use of a cycle count returns the first number of rows within the time range of the query. For more information, see -old-Using wwResolution, wwCycleCount, and wwRetrievalMode in the Same Query.

Also, the use of a time deadband and/or value deadband with delta retrieval produces differing results. For more information, see *Time Deadband (wwTimeDeadband)* on page 97 and *Value Deadband (wwValueDeadband)* on page 101.

Delta Retrieval - Query 2

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysTimeSec','SysTimeMin')
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:36'
         AND wwRetrievalMode = 'Delta'
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeSec	0

2001-12-09 11:35:00.000	SysTimeMin	35
2001-12-09 11:35:01.000	SysTimeSec	1
2001-12-09 11:35:02.000	SysTimeSec	2
2001-12-09 11:35:03.000	SysTimeSec	3
2001-12-09 11:35:04.000	SysTimeSec	4
...		
2001-12-09 11:35:58.000	SysTimeSec	58
2001-12-09 11:35:59.000	SysTimeSec	59
2001-12-09 11:36:00.000	SysTimeSec	0
2001-12-09 11:36:00.000	SysTimeMin	36

Delta Retrieval - Query 3

```
SELECT * FROM OpenQuery(INSQL,'SELECT DateTime, Value, Quality, QualityDetail
   FROM AnalogHistory
      WHERE TagName = "SysTimeSec"
         AND wwRetrievalMode = "Delta"
         AND Value = 10
         AND DateTime >="2001-07-27 03:00:00.000"
         AND DateTime <="2001-07-27 03:05:00.000"
      ')
```

The results are:

DateTime	Value	Quality	QualityDetail
2001-07-27 03:00:10.000	10	0	192
2001-07-27 03:01:10.000	10	0	192
2001-07-27 03:02:10.000	10	0	192
2001-07-27 03:03:10.000	10	0	192
2001-07-27 03:04:10.000	10	0	192

Delta Retrieval - Query 4

For a delta query, if both a wwCycleCount and a Value comparison are specified, the query will return the first number of rows (if available) that meet the value indicated.

```
SELECT * FROM OpenQuery(INSQL,'SELECT DateTime, Value, Quality, QualityDetail
   FROM AnalogHistory
      WHERE TagName = "SysTimeSec"
         AND wwRetrievalMode = "Delta"
         AND Value = 20
         AND wwCycleCount = 10
         AND DateTime >="2001-07-27 03:00:00.000"
         AND DateTime <="2001-07-27 03:20:00.000"
   ')
```

The results are:

DateTime	Value	Quality	QualityDetail

DateTime	Value	Quality	QualityDetail
2001-07-27 03:00:20.000	20	0	192
2001-07-27 03:01:20.000	20	0	192
2001-07-27 03:02:20.000	20	0	192
2001-07-27 03:03:20.000	20	0	192
2001-07-27 03:04:20.000	20	0	192
2001-07-27 03:05:20.000	20	0	192
2001-07-27 03:06:20.000	20	0	192
2001-07-27 03:07:20.000	20	0	192
2001-07-27 03:08:20.000	20	0	192
2001-07-27 03:09:20.000	20	0	192

Delta Retrieval - Initial Values

Initial values are special values that can be returned from queries that lie exactly on the query start time, even if there is not a data point that specifically matches the specified start time. If there is not a value exactly on the query start time, the last point before the start time will be returned with its DateTime set to the query start time and its Quality set to 133. If no value exists at or prior to the query start time, a NULL value will be returned at start time with QualityDetail set to 65536, OPCQuality set to 0, and Quality set to 1.

Querying the start time in exclusive form with the > operator indicates that a value should not be returned for the query start time if one does not exist. Querying the start time in inclusive form with the >= operator indicates that an initial value should be returned.

For example, the following exclusive query statement does not return an initial value for 2009-01-01 02:00:00.

```
DateTime > '2009-01-01 02:00:00'
```

However, the following inclusive query statement does return an initial value for 2009-01-01 02:00:00.

```
DateTime >= '2009-01-01 02:00:00'
```

No special final value is returned.

Delta Retrieval - Handling NULL Values

The initial NULL value after a non-NULL is always returned. Multiple NULL values are suppressed. The first non-NULL after a NULL is always returned even if it is the same as the previous non-NULL value.

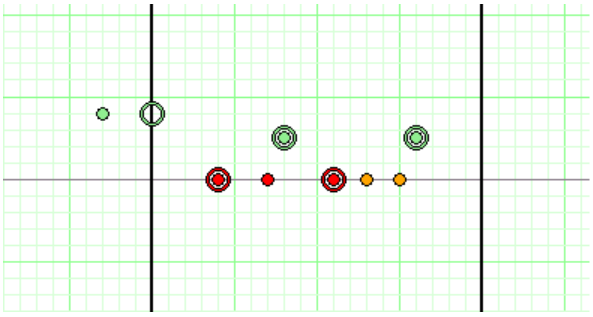
```
SELECT TagName, DateTime, Value, QualityDetail
   FROM History
      WHERE TagName = 'A001'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwRetrievalMode = 'Delta'
```

This query can be run against the following sample data:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:17	0.8	192
A001	2009-09-12 00:24	0.0	249

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:27	0.0	249
A001	2009-09-12 00:28	0.5	192
A001	2009-09-12 00:31	0.0	249
A001	2009-09-12 00:33	0.0	24
A001	2009-09-12 00:35	0.0	24
A001	2009-09-12 00:36	0.5	192

The following is a graphical representation of the data:



The results are:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:20	0.8	192
A001	2009-09-12 00:24	NULL	249
A001	2009-09-12 00:28	0.5	192
A001	2009-09-12 00:31	NULL	249
A001	2009-09-12 00:36	0.5	192

The sample data points and the results are mapped on the following chart. Only the data falling between the time start and end marks at 00:20 and 00:40 (shown on the chart as dark vertical lines) are returned by the query.

Because there is no value that matches the start time, an initial value at 00:20 is returned in the results based on the value of the preceding data point at 00:16. Because there is no change in the value at 00:27 from the value at 00:24, the data point appears on the chart but does not appear in the results. Similarly, the two 0.0 values at 00:33 and 00:35 are also excluded from the results. However, the non-NULL value at 00:36 is returned, even though it is the same as the value at 00:28, because it represents a delta from the preceding (NULL) value at 00:35.

Full Retrieval

In full retrieval mode, all stored data points are returned, regardless of whether a value or quality has changed since the last value. This mode allows the same value and quality pair (or NULL value) to be returned consecutively with their actual timestamps. It works with all types of tags.

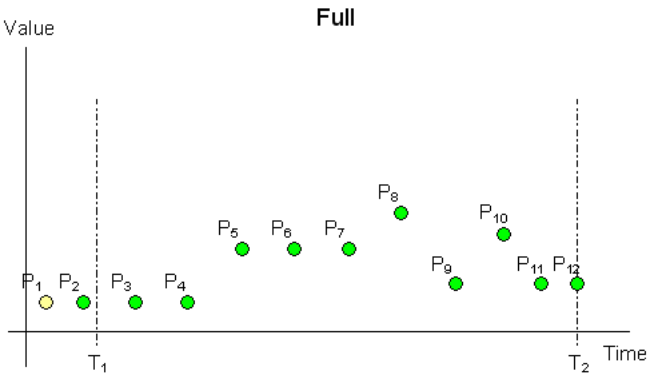
By using full retrieval in conjunction with storage without filtering (that is, no delta or cyclic storage mode is applied at the historian), you can retrieve all values that originated from the plant floor data source or from another application.

Full retrieval best represents the process measurements recorded by the AVEVA Historian. However, it creates a higher load for the server, the network and the client system because a very large number of records may be returned for longer time periods.

For full retrieval for replicated summary tags on a tier-2 historian, if a point with doubtful quality is returned as the result of a value selection from an input summary point with a contained gap, the same point can be returned again with good quality if the same value is selected again from the next input summary point that has good quality.

Full Retrieval - How It Works

The following illustration shows how values are returned for full retrieval:



Data is retrieved in full mode with a start time of T_1 and an end time of T_2. Each dot in the graphic represents an actual data point stored on the historian. From these points, the following are returned:

- P_2, because there is no actual data point at T_1
- P_3 through P_{12}, because they represent stored data points during the time period

Full Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in full retrieval mode. For more information, see the following sections:

- *History Version (wwVersion)* on page 104
- *Quality Rule (wwQualityRule)* on page 111

Full Retrieval - Query Example

For example, the following query uses full retrieval mode:

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysTimeSec','SysTimeMin')
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:36'
         AND wwRetrievalMode = 'Full'
```

Full Retrieval - Initial Values

Full retrieval mode handles initial values the same way as delta mode. For more information on initial values, see *Delta Retrieval - Initial Values* on page 34.

Interpolated Retrieval

Interpolated retrieval works like cyclic retrieval, except that interpolated values are returned if there is no actual data point stored at the cycle boundary.

This retrieval mode is useful if you want to retrieve cyclic data for slow-changing tags. For a trend, interpolated retrieval results in a smoother curve instead of a "stair-stepped" curve. This mode is also useful if you have a slow-changing tag and a fast-changing tag and want to retrieve data for both. Finally, some advanced applications require more evenly spaced values than would be returned if interpolation was not applied.

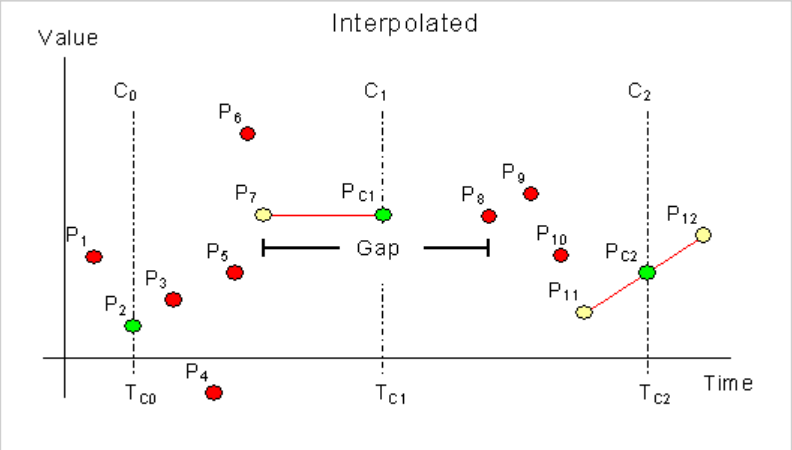
By default, interpolated retrieval uses the interpolation setting specified for the tag in the AVEVA Historian. This means that if a tag is set to use stair-step interpolation, interpolated retrieval gives the same results as cyclic retrieval.

Interpolation is only applied to analog tags. If you retrieve data for other types of tags, stair-step interpolation is used, and the results are the same as for cyclic retrieval.

Interpolated retrieval is a bit slower than cyclic retrieval. It shares the limitations of cyclic retrieval in that it may not accurately represent the stored process data.

Interpolated Retrieval - How It Works

The following illustration shows how the values for an analog tag that is configured for linear interpolation are returned when using interpolated retrieval.



Data is retrieved in interpolated mode with a start time of TC_0 and an end time of TC_2. The resolution has been set in such a way that the historian returns data for three cycle boundaries at TC_0, TC_1, and TC_2. P_1 to P_{12} represent actual data points stored on the historian. Of these points, eleven represent normal analog values, and one, P_7, represents a NULL value due to an I/O Server disconnect, which causes a gap in the data between P_7 and P_8.

The green points (P_2, PC_1, PC_2) are returned. The yellow points (P_7, P_{11}, P_{12}) are used to interpolate the returned value for each cycle. The red points are considered, but not used in calculating the points to return.

Because P_2 is located exactly at the query start time, it is returned at that time without the need for any interpolation. At the following cycle boundary, point PC1 is returned, which is the NULL value represented by P_7 shifted forward to time TC_1. At the last cycle boundary, point PC_2 is returned, which has been interpolated using points P_{11} and P_{12}.

You can use various parameters to adjust which values are returned in interpolated retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Interpolation Type (wwInterpolationType)* on page 105
- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111

Interpolated Retrieval - Query Examples

To use the interpolated mode, set the following parameter in your query.

```
wwRetrievalMode = 'Interpolated'
```

For examples, see the following:

- *Interpolated Retrieval - Query 1* on page 38
- *Interpolated Retrieval - Query 2* on page 39
- *Interpolated Retrieval - Query 3* on page 40

Interpolated Retrieval - Query 1

Two analog tags and a discrete tag are retrieved from the History table, using linear interpolation. The start and end times are offset to show interpolation of the SysTimeMin tag. The data points at all cycle boundaries are interpolated for the two analog tags, while the values returned for the discrete tag are stair-stepped.

```
SELECT DateTime, TagName, Value, wwInterpolationType FROM History
    WHERE TagName IN ('SysTimeMin', 'ReactTemp', 'SysPulse')
      AND DateTime >= '2005-04-11 12:02:30'
      AND DateTime <= '2005-04-11 12:06:30'
      AND wwRetrievalMode = 'Interpolated'
      AND wwInterpolationType = 'Linear'
      AND wwResolution = 60000
```

The results are:

DateTime	TagName	Value	wwInterpolationType
2005-04-11 12:02:30.000	SysTimeMin	2.5	LINEAR
2005-04-11 12:02:30.000	ReactTemp	23.2	LINEAR
2005-04-11 12:02:30.000	SysPulse	1.0	STAIRSTEP

DateTime	TagName	Value	wwInterpolationType
2005-04-11 12:03:30.000	SysTimeMin	3.5	LINEAR
2005-04-11 12:03:30.000	ReactTemp	139.96753	LINEAR
2005-04-11 12:03:30.000	SysPulse	0.0	STAIRSTEP
2005-04-11 12:04:30.000	SysTimeMin	4.5	LINEAR
2005-04-11 12:04:30.000	ReactTemp	111.49636	LINEAR
2005-04-11 12:04:30.000	SysPulse	1.0	STAIRSTEP
2005-04-11 12:05:30.000	SysTimeMin	5.5	LINEAR
2005-04-11 12:05:30.000	ReactTemp	17.00238	LINEAR
2005-04-11 12:05:30.000	SysPulse	0.0	STAIRSTEP
2005-04-11 12:06:30.000	SysTimeMin	6.5	LINEAR
2005-04-11 12:06:30.000	ReactTemp	168.99531	LINEAR
2005-04-11 12:06:30.000	SysPulse	1.0	STAIRSTEP

Interpolated Retrieval - Query 2

If you omit the interpolation type in the query, the historian determines which interpolation type to use for an analog tag based on the value of the InterpolationType column in the AnalogTag table, in conjunction with the InterpolationTypeInteger and InterpolationTypeReal system parameters.

In the following query both analog tags are set to use the system default through the AnalogTag table, while the InterpolationTypeInteger and InterpolationTypeReal system parameters are set to 0 and 1, respectively. Because SysTimeMin is defined as a 2-byte integer and ReactTemp is defined as a real we see that only rows for ReactTemp are interpolated.

```
SELECT DateTime, TagName, Value, wwInterpolationType FROM History
   WHERE TagName IN ('SysTimeMin', 'ReactTemp', 'SysPulse')
      AND DateTime >= '2005-04-11 12:02:30'
      AND DateTime <=  '2005-04-11 12:06:30'
      AND wwRetrievalMode = 'Interpolated'
      AND wwResolution = 60000
```

The results are:

DateTime	TagName	Value	wwInterpolationType
2005-04-11 12:02:30.000	SysTimeMin	2.0	STAIRSTEP
2005-04-11 12:02:30.000	ReactTemp	23.2	LINEAR
2005-04-11 12:02:30.000	SysPulse	1.0	STAIRSTEP
2005-04-11 12:03:30.000	SysTimeMin	3.0	STAIRSTEP
2005-04-11 12:03:30.000	ReactTemp	139.96753	LINEAR
2005-04-11 12:03:30.000	SysPulse	0.0	STAIRSTEP
2005-04-11 12:04:30.000	SysTimeMin	4.0	STAIRSTEP
2005-04-11 12:04:30.000	ReactTemp	111.49636	LINEAR
2005-04-11 12:04:30.000	SysPulse	1.0	STAIRSTEP
2005-04-11 12:05:30.000	SysTimeMin	5.0	STAIRSTEP

DateTime	TagName	Value	wwInterpolationType
2005-04-11 12:05:30.000	ReactTemp	17.00238	LINEAR
2005-04-11 12:05:30.000	SysPulse	0.0	STAIRSTEP
2005-04-11 12:06:30.000	SysTimeMin	6.0	STAIRSTEP
2005-04-11 12:06:30.000	ReactTemp	168.99531	LINEAR
2005-04-11 12:06:30.000	SysPulse	1.0	STAIRSTEP

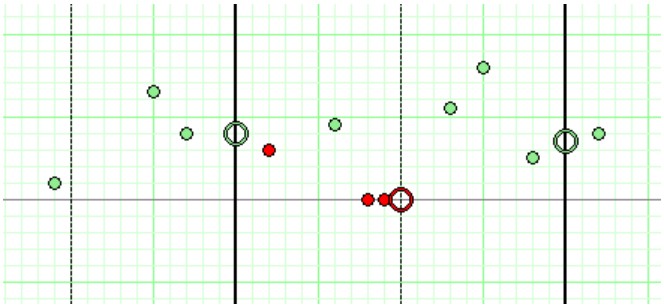
Interpolated Retrieval - Query 3

```
SELECT TagName, DateTime, Value, QualityDetail, wwInterpolationType
   FROM History
      WHERE TagName = 'A001'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwRetrievalMode = 'Interpolated'
         AND wwResolution = '10000'
```

This query can be run against the following sample data:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:09	0.2	192
A001	2009-09-12 00:15	1.3	192
A001	2009-09-12 00:17	0.8	192
A001	2009-09-12 00:22	0.6	249
A001	2009-09-12 00:26	0.9	192
A001	2009-09-12 00:28	0.0	249
A001	2009-09-12 00:29	0.0	249
A001	2009-09-12 00:33	1.1	192
A001	2009-09-12 00:35	1.6	192
A001	2009-09-12 00:38	0.5	192
A001	2009-09-12 00:42	0.8	192

The following is a graphical representation of the data:



The results are:

Tagname	DateTime	Value	QualityDetail	wwInterpolationType
A001	2009-09-12 00:20	0.8	192	STAIRSTEP
A001	2009-09-12 00:30	NULL	249	STAIRSTEP
A001	2009-09-12 00:40	0.5	192	LINEAR

The sample data points and the results are mapped on the following chart. Only the data falling between the time start and end marks at 00:20 and 00:40 (shown on the chart as dark vertical lines) are returned by the query.

Because there is no value that matches the start time, an initial value at 00:20 is returned in the results based on the preceding data point at 00:17 because the following data point at 00:22 is NULL. Because a NULL value precedes the 00:30 cycle boundary at 00:29, the NULL is returned at the cycle boundary. The value at 00:40 is an interpolation of the data points at 00:38 and 00:42.

Interpolated Retrieval - Initial and Final Values

A value is returned at the start time and end time of the query using interpolation of the surrounding points.

Interpolated Retrieval - Handling NULL Values

When a NULL value precedes a cycle boundary, that NULL will be returned at the cycle boundary.

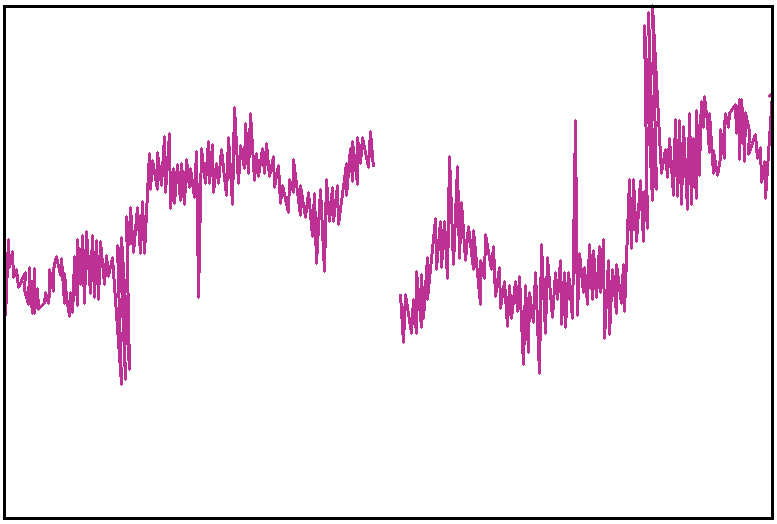
If a valid value precedes a cycle boundary, but is followed by a NULL value after the cycle boundary, no interpolation will be used and wwInterpolationType will be set to STAIRSTEP for that value.

Best Fit Retrieval

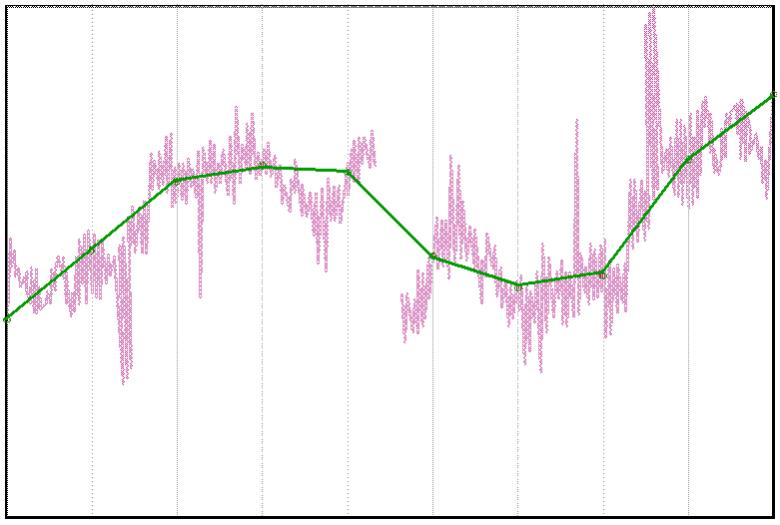
For the " best fit" retrieval mode, the total time for the query is divided into even sub-periods, and then up to five values are returned for each sub-period:

- First value in the period
- Last value in the period
- Minimum value in the period, with its actual time
- Maximum value in the period, with its actual time
- The first "exception" in the period (non-Good quality)

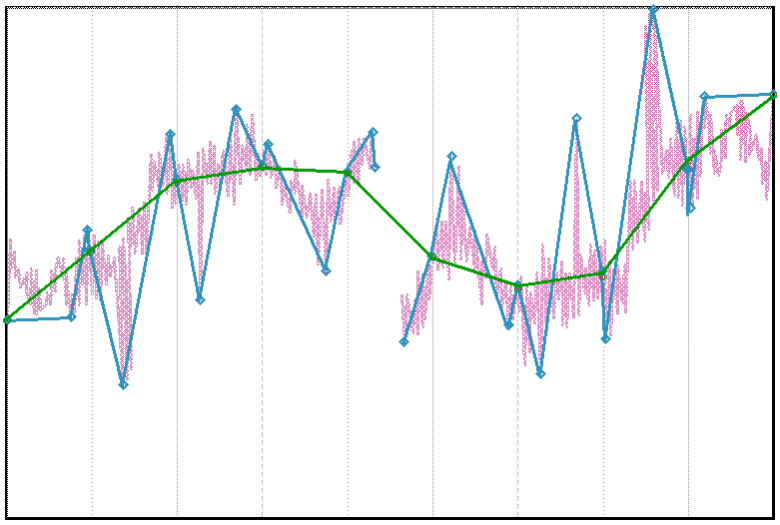
"Best fit" retrieval allows for a compromise between delta retrieval and cyclic retrieval. For example, delta retrieval can accurately represent a process over a long period of time, as shown in the following trend. However, to achieve this representation, a large number of data values must be returned.



If cyclic retrieval is used to retrieve the data, the retrieval is much more efficient, because fewer values are returned. However, the representation is not as accurate, as the following trend shows.



"Best fit" retrieval allows for faster retrieval, as typically achieved by using cyclic retrieval, plus the better representation typically achieved by using delta retrieval. This is shown in the following trend.



For example, for one week of five-second data, 120,960 values would be returned for delta retrieval, versus around 300 values for best-fit retrieval.

Best-fit retrieval uses retrieval cycles, but it is not a true cyclic mode. Apart from the initial value, it only returns actual delta points. For example, if one point is both the first value and the minimum value in a cycle, it is returned only one time. In a cycle where a tag has no points, nothing is returned.

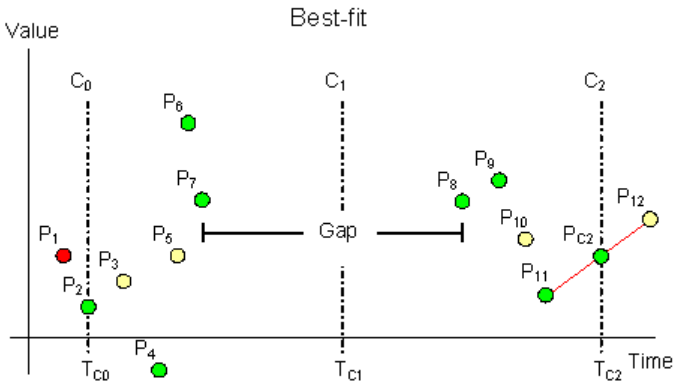
As in cyclic retrieval, the number of cycles is based on the specified resolution or cycle count. However, the number of values returned is likely to be more than one per cycle.

All points are returned in chronological order. If multiple points are to be returned for a particular timestamp, then those points are returned in the order in which the corresponding tags were specified in the query.

The best-fit algorithm is only applied to analog and analog summary tags. For all other tags, delta results are returned.

Best Fit Retrieval - How It Works

The following illustration shows how the best-fit algorithm selects points for an analog tag.



Data is retrieved in best-fit mode with a start time of T_{C0} and an end time of T_{C2}. The resolution has been set in such a way that the historian returns data for two complete cycles starting at T_{C0} and T_{C1} and an incomplete cycle starting at T_{C2}. P_1 to P_{12} represent actual data points stored on the historian. Of these points, eleven represent normal analog values, and one, P_7, represents a NULL value due to an I/O Server disconnect, which causes a gap in the data between P_7 and P_8.

Because P_2 is located exactly at the start time, no initial value needs to be interpolated at the start time. Therefore, point P_1 is not considered at all. All other points are considered, but only the points indicated by green markers on the graph are returned.

From the first cycle, four points are returned:

- P_2 as the initial value of the query, as well as the first value in the cycle
- P_4 as the minimum value in the cycle
- P_6 as both the maximum value and the last value in the cycle
- P_7 as the first (and only) occurring exception in the cycle

From the second cycle, three points are returned:

- P_8 as the first value in the cycle
- P_9 as the maximum value in the cycle
- P_{11} as both the minimum value and the last value in the cycle
- As no exception occurs in the second cycle, none is returned.

Because the tag does not have a point exactly at the query end time, where an incomplete third cycle starts, the end value P_{C2} is interpolated between P_{11} and P_{12}, assuming that linear interpolation is used.

Best Fit Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in best-fit retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Interpolation Type (wwInterpolationType)* on page 105
- *Quality Rule (wwQualityRule)* on page 111

Best Fit Retrieval - Query Example

To use the best fit retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'BestFit'
```

For example, an analog tag is retrieved over a five-minute period using the best-fit retrieval mode. The wwResolution parameter is set to 60000, thus specifying five 1-minute cycles. Within each cycle, the retrieval sub-system returns the first, minimum, maximum, and last data points. There are no exception (NULL) points in the time period. Notice how the points at the query start time and at the query end time are interpolated, while all other points are actual delta points.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 1), Value) AS Value,
wwInterpolationType AS IT FROM History
      WHERE TagName = 'ReactTemp'
```

```
AND DateTime >= '2005-04-11 12:15:00'
AND DateTime <= '2005-04-11 12:20:00'
AND wwRetrievalMode = 'BestFit'
AND wwResolution = 60000
```

The results are:

	DateTime	TagName	Value	IT
(initial, first, min)	2005-04-11 12:15:00.000	ReactTemp	40.7	LINEAR
(max in interval 1)	2005-04-11 12:15:38.793	ReactTemp	196.0	STAIRSTEP
(last in interval 1)	2005-04-11 12:15:58.810	ReactTemp	159.2	STAIRSTEP
(first, max in interval 2)	2005-04-11 12:16:00.013	ReactTemp	156.9	STAIRSTEP
(last, min in interval 2)	2005-04-11 12:16:58.857	ReactTemp	16.3	STAIRSTEP
(first, min in interval 3)	2005-04-11 12:17:00.060	ReactTemp	14.0	STAIRSTEP
(last, max in interval 3)	2005-04-11 12:17:58.793	ReactTemp	151.0	STAIRSTEP
(first in interval 4)	2005-04-11 12:18:00.107	ReactTemp	156.0	STAIRSTEP
(max in interval 4)	2005-04-11 12:18:10.057	ReactTemp	196.0	STAIRSTEP
(last, min in interval 4)	2005-04-11 12:18:58.837	ReactTemp	106.3	STAIRSTEP
(first, max in interval 5)	2005-04-11 12:19:00.040	ReactTemp	104.0	STAIRSTEP
(min in interval 5)	2005-04-11 12:19:31.320	ReactTemp	14.0	STAIRSTEP
(last in interval 5)	2005-04-11 12:19:58.773	ReactTemp	26.0	STAIRSTEP
(end bounding value)	2005-04-11 12:20:00.000	ReactTemp	30.7	LINEAR

Best Fit Retrieval - Initial and Final Values

A point will be returned at the query start time and the query end time for each tag queried, if a point exists for that tag at or after the end time of the query. The values of the initial and final points will be determined by interpolating the points preceding and following the query start or query end time.

Standard interpolation rules will be used to return the initial and final values. For more information, see *Interpolated Retrieval* on page 37.

Best Fit Retrieval - Handling NULL Values

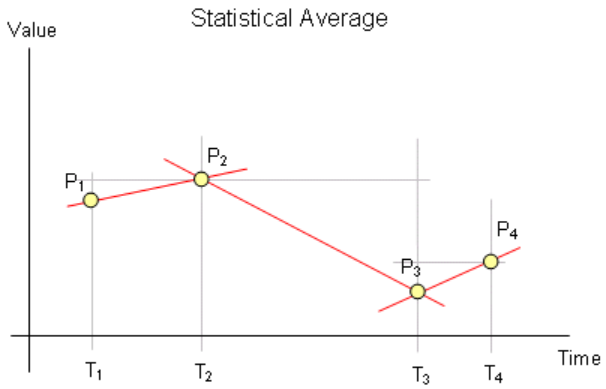
When any of the four good points are returned from a cycle that contains gaps or from an incomplete cycle with the query end time located inside of the calculation cycle the quality detail of each of the non-null points returned is modified to alert the user to this fact. This is done by performing a logical OR operation of the value 4096, which means partial cycle, onto the existing quality detail. (This is the delta point equivalent to the use of PercentGood for cyclic.)

Average Retrieval

For the time-weighted average (in short: "average") retrieval mode, a time-weighted average algorithm is used to calculate the value to be returned for each retrieval cycle.

For a statistical average, the actual data values are used to calculate the average. The average is the sum of the data values divided by the number of data values. For the following data values, the statistical average is computed as:

$(P_1 + P_2 + P_3 + P_4) / 4) = \text{Average}$

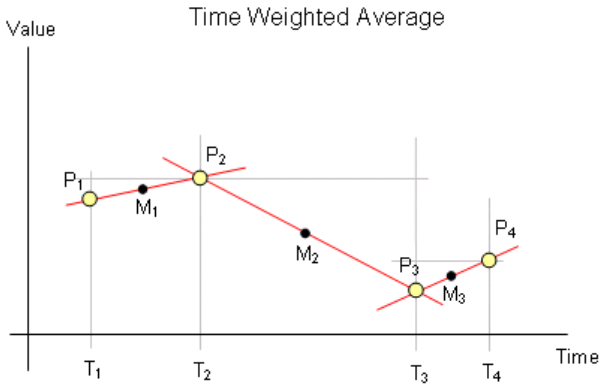


For a time-weighted average, values are multiplied by the time difference between the points to determine the time-weighted value. Therefore, the longer a tag had a particular value, the more weight that value holds in the overall average. The overall average is determined by adding all of the time-weighted values and then dividing that number by the total amount of time.

Which values are weighted depends on the interpolation setting of the tag. For a tag that uses linear interpolation, the midpoints between values are weighted. For a tag that uses stair-step interpolation, the earlier of two values is weighted.

For the following data values of a tag that uses linear interpolation, the time-weighted average is computed as:

$$(((P_1 + P_2) / 2) \times (T_2 - T_1)) + (((P_2 + P_3) / 2) \times (T_3 - T_2)) + (((P_3 + P_4) / 2) \times (T_4 - T_3)) / (T_4 - T_1) = \text{Average}$$



If the same tag uses stair-step interpolation, the time-weighted average is:

$$((P_1 \times (T_2 - T_1)) + (P_2 \times (T_3 - T_2)) + (P_3 \times (T_4 - T_3))) / (T_4 - T_1) = \text{Average}$$

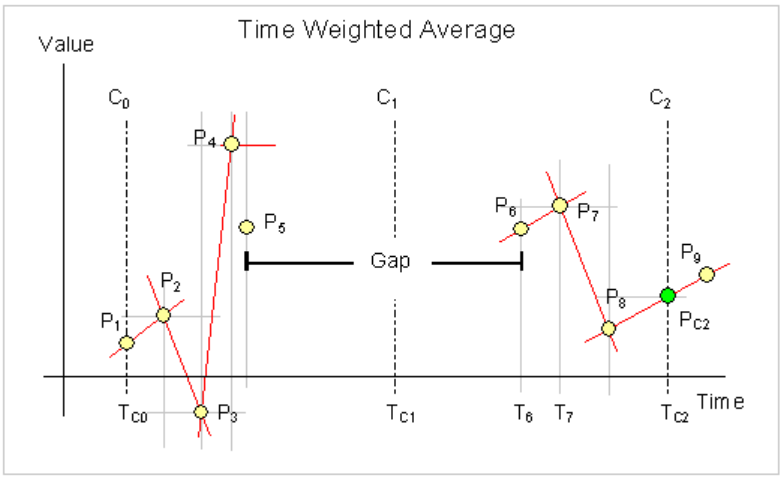
The SQL Server AVG aggregate is a simple statistical average. Using the average retrieval mode with a cycle count of 1 returns data much faster than the AVG aggregate, and usually more accurately due to the time weighting. The Event subsystem also returns a simple statistical average.

Average retrieval returns one row for each tag in the query for each cycle. The number of cycles is based on the specified resolution or cycle count.

The time-weighted average algorithm is only applied to analog and analog summary tags. If you use average retrieval with other tags, the results are the same as when using regular cyclic retrieval.

Average Retrieval - How It Works

The following illustration shows how the time-weighted average is calculated for an analog tag that uses linear interpolation.



Data is retrieved in average mode with a start time of T_{C0} and an end time of T_{C2}. The resolution has been set in such a way that the historian returns data for two complete cycles starting at T_{C0} and T_{C1} and an incomplete cycle starting at T_{C2}. P_1 to P_9 represent actual data points stored on the historian. Of these points, eight represent normal analog values, and one, P_5, represents a NULL due to an I/O Server disconnect, which causes a gap in the data between P_5 and P_6. Assume that the query calls for timestamping at the end of the cycle.

Results are calculated as follows:

- The "initial value" returned at the query start time (T_{C0}) is the time-weighted average of the points in the last cycle preceding T_{C0}.
- The value returned at T_{C1} is the time-weighted average of the points in the cycle starting at T_{C0}.
- The value returned at the query end time (T_{C2}) is the time-weighted average of the points in the cycle starting at T_{C1}.

To understand how the time-weighted average is calculated, observe the last cycle as an example. First, the area under the curve must be calculated. This curve is indicated by the red line through P_6, P_7, P_8 and P_{C2}, where P_{C2} represents the interpolated value at time T_{C2} using points P_8 and P_9. The data gap caused by the I/O Server disconnect does not contribute anything to this area. If a quality rule of "good" has been specified, then points with doubtful quality will not contribute anything to the area, either.

To understand how the area is calculated, consider points P_6 and P_7. The area contribution between these two points is calculated by multiplying the arithmetic average of value P_6 and value P_7 by the time difference between the two points. The formula is:

$((P_6 + P_7) / 2) \times (T_7 - T_6)$

When the area for the whole cycle has been calculated, the time-weighted average is calculated by dividing that area by the cycle time, less any periods within the cycle that did not contribute anything to the area calculation. The result is returned at the cycle end time.

If you take a closer look at points P_4 and P_5 in the example, you can see that the red line through point P_4 is parallel to the x-axis. This is because P_5 represents a NULL, which cannot be used to calculate an arithmetic average. Instead, the value P_4 is used for the whole time period between points P_4 and P_5.

The area calculation is signed. If the arithmetic average between two points is negative, then the contribution to the area is negative.

Average Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in average retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Interpolation Type (wwInterpolationType)* on page 105
- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111

Average Retrieval - Query Examples

To use the average mode, set the following parameter in your query.

```
wwRetrievalMode = 'Average'
```

For examples, see the following:

- *Average Retrieval - Query 1* on page 49
- *Average Retrieval - Query 2* on page 49

For an additional example, see Querying Aggregate Data in Different Ways.

Average Retrieval - Query 1

The time-weighted average is computed for each of five 1-minute long cycles.

Note that the wwTimeStampRule parameter is set to "Start" in the query. This means that the value stamped at 11:18:00.000 represents the average for the interval 11:18 to 11:19, the value stamped at 11:19:00.000 represents the average for the interval 11:19 to 11:20, and so on. If no timestamp rule is specified in the query, then the default setting in the TimeStampRule system parameter is used.

In the first cycle there are no points, so a NULL is returned. In the second cycle value points are found covering 77.72 percent of the time as returned in PercentGood. This means that the returned average is calculated based on 77.72 percent of the cycle time. Because the same OPCQuality is not found for all the points in the cycle, OPCQuality is set to Doubtful. In the remaining three cycles, only good points occur, all with an OPCQuality of 192.

Because no quality rule is specified in the query using the wwQualityRule parameter, the query uses the default as specified by the QualityRule system parameter. If a quality rule of Extended is specified, any point stored with doubtful OPCQuality will be used to calculate the average, and the point time will therefore be included in the calculation of PercentGood.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Value, OPCQuality,
PercentGood FROM History
   WHERE TagName = 'ReactTemp'
      AND DateTime >= '2005-04-11 11:18:00'
      AND DateTime <  '2005-04-11 11:23:00'
      AND wwRetrievalMode = 'Average'
      AND wwCycleCount = 5
      AND wwTimeStampRule = 'Start'
```

The results are:

	DateTime	TagName	Value	OPCQuality	PercentGood
(cycle 1)	2005-04-11 11:18:00.000	ReactTemp	NULL	0	0.0
(cycle 2)	2005-04-11 11:19:00.000	ReactTemp	70.00	64	77.72
(cycle 3)	2005-04-11 11:20:00.000	ReactTemp	153.99	192	100.0
(cycle 4)	2005-04-11 11:21:00.000	ReactTemp	34.31	192	100.0
(cycle 5)	2005-04-11 11:22:00.000	ReactTemp	134.75	192	100.0

Average Retrieval - Query 2

This query demonstrates the use of the average retrieval mode in a wide query. Time-weighted average values are returned for the analog tags ReactTemp and ReactLevel, while regular cyclic points are returned for the discrete tag, WaterValve.

```
SELECT * FROM OpenQuery(INSQL,
   'SELECT DateTime, ReactTemp, ReactLevel, WaterValve FROM WideHistory
      WHERE DateTime >= "2004-06-07 08:00"
         AND DateTime < "2004-06-07 08:05"
         AND wwRetrievalMode = "Average"
```

```
        AND wwCycleCount = 5
')
```

The results are:

DateTime	ReactTemp	ReactLevel	WaterValve
2004-06-07 08:00:00.000	47.71621	1676.69716	1
2004-06-07 08:01:00.000	157.28076	1370.88097	0
2004-06-07 08:02:00.000	41.33734	797.67296	1
2004-06-07 08:03:00.000	122.99525	1921.66771	0
2004-06-07 08:04:00.000	105.28866	606.40488	1

Average Retrieval - Initial and Final Values

If wwTimeStampRule = END, the initial value is calculated by performing an average calculation on the cycle leading up to the query start time. No special handling is done for the final value.

If wwTimeStampRule = START, the final value is calculated by performing an average calculation on the cycle following the query end time. No special handling is done for the initial value.

Average Retrieval - Handling NULL Values

Gaps introduced by NULL values are not included in the average calculations. The average only considers the time ranges with good values. TimeGood indicates the total time per cycle that the tags value was good.

Minimum Retrieval

The minimum value retrieval mode returns the minimum value from the actual data values within a retrieval cycle. If there are no actual data points stored on the historian for a given cycle, nothing is returned. NULL is returned if the cycle contains one or more NULL values.

As in cyclic retrieval, the number of cycles is based on the specified resolution or cycle count. However, minimum retrieval is not a true cyclic mode. Apart from the initial value, all points returned are delta points.

Minimum retrieval only works with analog tags. For all other tags, normal delta results are returned.

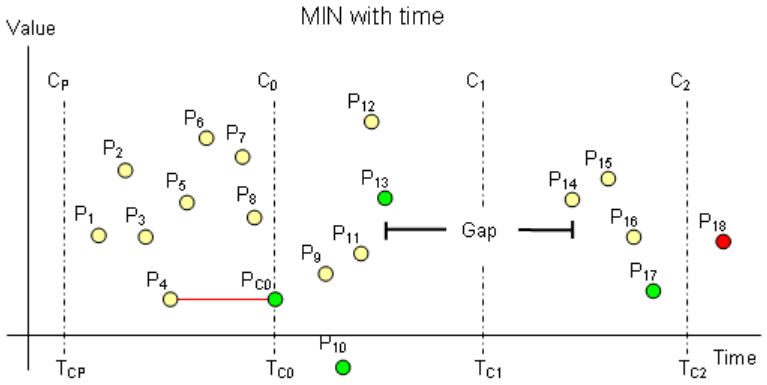
All returned values are in chronological order. If multiple points are to be returned for a particular timestamp, they are returned in the order in which the tags were specified in the query. If the minimum value occurs several times in a cycle, the minimum value with the earliest timestamp is returned.

The minimum retrieval mode must use the "<=" operator for the ending date/time.

Using the minimum retrieval mode with a cycle count of 1 returns the same results as the SQL Server MIN aggregate; however, the data is returned much faster.

Minimum Retrieval - How It Works

The following illustration shows how the minimum value is selected for an analog tag.



This example has a start time of T_{C0} and an end time of T_{C2}. The resolution has been set in such a way that the historian returns data for two complete cycles starting at T_{C0} and T_{C1}, a "phantom" cycle starting at T_{CP}, and an incomplete cycle starting at T_{C2}. The phantom cycle has the same duration as the first cycle in the query period, extending back in time from the query start time.

For the queried tag, a total of 18 points are found throughout the cycles, represented by the markers P_1 through P_{18}. Of these points, 17 represent normal analog values. The point P_{13} represents a NULL due to an I/O Server disconnect, which causes a gap in the data between P_{13} and P_{14}.

The minimum value for the "phantom" cycle starting at T_{CP} is returned as the initial value at T_{C0}. Point P_{18} is not considered at all because it is outside of the query time frame. All other points are considered, but only the points indicated by green markers on the graph are returned (P_{10}, P_{13}, and P_{17}).

In total, four points are returned:

- P_4 as the minimum value of the "phantom" cycle and the initial point
- P_{10} as the minimum value in the first cycle
- P_{13} as the first and only exception occurring in the first cycle
- P_{17} as the minimum value in the second cycle

No points are returned for the incomplete third cycle starting at the query end time, because the tag does not have a point exactly at that time.

If the minimum value of the first cycle is located exactly at the query start time, both this value and the minimum value of the phantom cycle are returned.

Minimum Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in minimum retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Quality Rule (wwQualityRule)* on page 111

Minimum Retrieval - Query Examples

To use the minimum mode, set the following parameter in your query:

```
wwRetrievalMode = 'Min'
```

or

```
wwRetrievalMode = 'Minimum'
```

For examples, see the following:

- *Minimum Retrieval - Query 1* on page 52
- *Minimum Retrieval - Query 2* on page 52
- *Minimum Retrieval - Query 3* on page 53

Minimum Retrieval - Query 1

In this example, an analog tag is retrieved over a five minute period, using the minimum retrieval mode. Because the wwResolution parameter is set to 60000, each cycle is exactly one minute long. The minimum data value is returned from each of these cycles.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Value FROM History
    WHERE TagName = 'ReactTemp'
    AND DateTime >= '2005-04-11 11:21:00'
    AND DateTime <= '2005-04-11 11:26:00'
    AND wwRetrievalMode = 'Minimum'
    AND wwResolution = 60000
```

The initial value at the query start time is the minimum value found in the phantom cycle before the start time of the query.

The results are:

	DateTime	TagName	Value
(phantom cycle)	2005-04-11 11:21:00.000	ReactTemp	104.00
(cycle 1)	2005-04-11 11:21:30.837	ReactTemp	14.00
(cycle 2)	2005-04-11 11:22:00.897	ReactTemp	36.00
(cycle 3)	2005-04-11 11:23:59.567	ReactTemp	18.60
(cycle 4)	2005-04-11 11:24:02.083	ReactTemp	14.00
(cycle 5)	2005-04-11 11:25:59.550	ReactTemp	108.60

Minimum Retrieval - Query 2

In this example, the minimum retrieval mode is used in a manner equivalent to using the SQL Server MIN aggregate. Note that the cycle producing the result is the five-minute phantom cycle just before the query start time.

```
SELECT TOP 1 DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Value FROM
History
    WHERE TagName = 'ReactTemp'
        AND DateTime >= '2005-04-11 11:31:00'
        AND DateTime <= '2005-04-11 11:31:00'
        AND wwRetrievalMode = 'Minimum'
        AND wwResolution = 300000
```

The results are:

	DateTime	TagName	Value
(phantom cycle)	2005-04-11 11:31:00.000	ReactTemp	14.00

Minimum Retrieval - Query 3

This example shows how the minimum retrieval mode marks the QualityDetail column to indicate that a minimum value is returned based on an incomplete cycle. In this case, an incomplete cycle is a cycle that either contained periods with no values stored or a cycle that was cut short because the query end time was located inside the cycle. All values returned for the QualityDetail column are documented in the QualityMap table in the Runtime database.

```
SELECT DateTime, TagName, Value, QualityDetail FROM History
   WHERE TagName = 'SysTimeSec'
      AND DateTime >= '2005-04-11 11:18:50'
      AND DateTime <= '2005-04-11 11:20:50'
      AND wwRetrievalMode = 'Minimum'
      AND wwResolution = 60000
```

The results are:

	DateTime	TagName	Value	QualityDetail
(phantom cycle)	2005-04-11 11:18:50.000	SysTimeSec	NULL	65536
(cycle 1)	2005-04-11 11:19:13.000	SysTimeSec	13.0	4140
(cycle 2)	2005-04-11 11:20:00.000	SysTimeSec	0.0	192
(cycle 3)	2005-04-11 11:20:50.000	SysTimeSec	50.0	4288

Minimum Retrieval - Initial and Final Values

For analog tags, the minimum value of the tag in the cycle leading up to the query start time is returned with its timestamp changed to the query start time. If there is no point exactly at the "phantom" cycle start time, the point leading up to the phantom cycle is also considered for the minimum calculation. (No adjustments are made to the quality of the initial point even though the timestamp may have been altered.) Apart from the initial value, all points returned are delta points. (For more information on initial values, see *Delta Retrieval - Initial Values* on page 34.)

If a point occurs exactly on the query end time, that point will be returned with the partial cycle bit, 4096, set in quality detail. If there is more than one such point, only the first point will be returned.

Minimum Retrieval - Handling NULL Values and Incomplete Cycles

The first NULL value in a cycle is returned.

When a minimum value is returned from a cycle that contains gaps (including a gap extended from the previous cycle) or from an incomplete cycle with the query end time located inside of the calculation cycle, the point's quality detail is modified to flag this. This is done by performing a logical OR operation of the value 4096, which indicates a partial cycle, onto the existing quality detail.

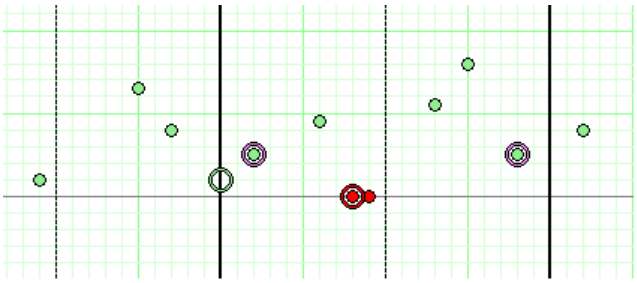
As an example of how minimum retrieval mode handles NULLs, consider the following query:

```
SELECT TagName, DateTime, Value, QualityDetail
    FROM History
        WHERE TagName = 'A001'
            AND DateTime >= '2009-09-12 00:20'
            AND DateTime <= '2009-09-12 00:40'
            AND wwResolution = 10000
            AND wwRetrievalMode = 'Minimum'
```

This query can be run against the following sample data:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:09	0.2	192
A001	2009-09-12 00:15	1.3	192
A001	2009-09-12 00:17	0.8	192
A001	2009-09-12 00:22	0.5	192
A001	2009-09-12 00:26	0.9	192
A001	2009-09-12 00:28	0.0	249
A001	2009-09-12 00:29	0.0	249
A001	2009-09-12 00:33	1.1	192
A001	2009-09-12 00:35	1.6	192
A001	2009-09-12 00:38	0.5	192
A001	2009-09-12 00:42	0.8	192

The following is a graphical representation of the data:



The results are:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:20	0.2	192
A001	2009-09-12 00:22	0.5	4288
A001	2009-09-12 00:28	NULL	249
A001	2009-09-12 00:38	0.5	4288

The sample data points and the results are mapped on the following chart. Only the data falling between the time start and end marks at 00:20 and 00:40 (shown on the chart as dark vertical lines) are returned by the query. The resolution is set at 10,000 milliseconds.

Because there is no value that matches the start time, an initial value at 00:20 is returned based on the minimum value of the preceding cycle, which is the data point at 00:09. In the two subsequent cycles, the minimum values are at 00:22 and 00:38. The quality for these two values is set to 4288 (4096 + 192). The remaining data points are excluded because they are not minimums. In addition, the first NULL at 00:28 is included, but the second NULL (at 00:29) is not.

Maximum Retrieval

The maximum value retrieval mode returns the maximum value from the actual data values within a retrieval cycle. If there are no actual data points stored on the historian for a given cycle, nothing is returned. NULL is returned if the cycle contains one or more NULL values.

As in cyclic retrieval, the number of cycles is based on the specified resolution or cycle count. However, maximum retrieval is not a true cyclic mode. Apart from the initial value, all points returned are delta points.

Maximum retrieval only works with analog tags. For all other tags, normal delta results are returned.

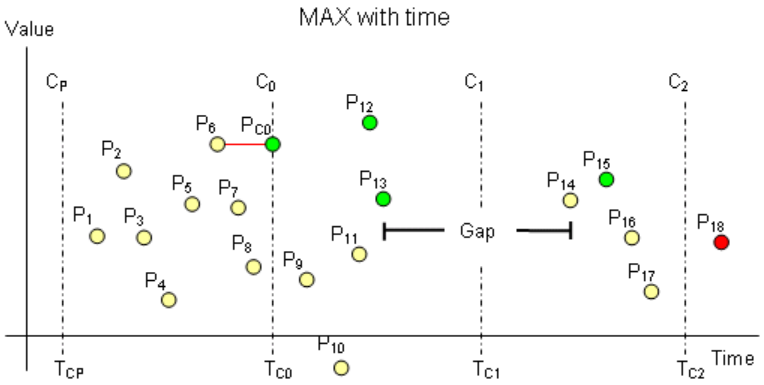
All returned values are in chronological order. If multiple points are to be returned for a particular timestamp, they are returned in the order in which the tags were specified in the query. If the maximum value occurs several times in a cycle, the maximum value with the earliest timestamp is returned.

The maximum retrieval mode must use the "<=" operator for the ending date/time.

Using the maximum retrieval mode with a cycle count of 1 returns the same results as the SQL Server MAX aggregate; however, the data is returned much faster.

Maximum Retrieval - How It Works

The following illustration shows how the maximum value is selected for an analog tag.



This example has a start time of T_{C0} and an end time of T_{C2}. The resolution has been set in such a way that the historian returns data for two complete cycles starting at T_{C0} and T_{C1}, a "phantom" cycle starting at T_{CP}, and an incomplete cycle starting at T_{C2}. The phantom cycle has the same duration as the first cycle in the query period, extending back in time from the query start time.

For the queried tag, a total of 18 points are found throughout the cycles, represented by the markers P_1 through P_{18}. Of these points, 17 represent normal analog values. The point P_{13} represents a NULL due to an I/O Server disconnect, which causes a gap in the data between P_{13} and P_{14}.

The maximum value for the "phantom" cycle starting at T_{CP} is returned as the initial value at T_{C0}. Point P_{18} is not considered at all because it is outside of the query time frame. All other points are considered, but only the points indicated by green markers on the graph are returned (P_{12}, P_{13}, and P_{15}).

In total, four points are returned:

- P_6 as the maximum value of the "phantom" cycle and the initial point
- P_{12} as the maximum value in the first cycle
- P_{13} as the first and only exception occurring in the first cycle
- P_{15} as the maximum value in the second cycle

No points are returned for the incomplete third cycle starting at the query end time, because the tag does not have a point exactly at that time.

If the maximum value of the first cycle is located exactly at the query start time, this value and the maximum value of the phantom cycle are returned.

Maximum Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in maximum retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Quality Rule (wwQualityRule)* on page 111

Maximum Retrieval - Query Examples

To use the maximum mode, set the following parameter in your query:

```
wwRetrievalMode = 'Max'
```

or

```
wwRetrievalMode = 'Maximum'
```

For examples, see the following:

- *Maximum Retrieval - Query 1* on page 56
- *Maximum Retrieval - Query 2* on page 57
- *Maximum Retrieval - Query 3* on page 57

Maximum Retrieval - Query 1

In this example, an analog tag is retrieved over a five-minute period, using the maximum retrieval mode. Because the wwResolution parameter is set to 60000, each cycle is exactly one minute long. The maximum data value is returned from each of these cycles.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Value FROM History
   WHERE TagName = 'ReactTemp'
      AND DateTime >= '2005-04-11 11:21:00'
      AND DateTime <= '2005-04-11 11:26:00'
      AND wwRetrievalMode = 'Maximum'
```

```
        AND wwResolution = 60000
```

The initial value at the query start time is the maximum value found in the phantom cycle before the start time of the query.

The results are:

Cycle	DateTime	TagName	Value
(phantom cycle)	2005-04-11 11:21:00.000	ReactTemp	196.00
(cycle 1)	2005-04-11 11:21:00.853	ReactTemp	101.70
(cycle 2)	2005-04-11 11:22:40.837	ReactTemp	196.00
(cycle 3)	2005-04-11 11:23:00.833	ReactTemp	159.20
(cycle 4)	2005-04-11 11:24:59.613	ReactTemp	146.00
(cycle 5)	2005-04-11 11:25:12.083	ReactTemp	196.00

Maximum Retrieval - Query 2

In this example, the maximum retrieval mode is used in a manner equivalent to using the SQL Server MIN aggregate. Note that the cycle producing the result is the five-minute phantom cycle just before the query start time.

```
SELECT TOP 1 DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Value FROM
History
   WHERE TagName = 'ReactTemp'
      AND DateTime >= '2005-04-11 11:31:00'
      AND DateTime <= '2005-04-11 11:31:00'
      AND wwRetrievalMode = 'Maximum'
      AND wwResolution = 300000
```

The results are:

	DateTime	TagName	Value
(phantom cycle)	2005-04-11 11:31:00.000	ReactTemp	196.00

Maximum Retrieval - Query 3

This example shows how the maximum retrieval mode marks the QualityDetail column to indicate that a maximum value is returned based on an incomplete cycle. In this case, an incomplete cycle is a cycle that either contained periods with no values stored or a cycle that was cut short because the query end time was located inside the cycle. All values returned for the QualityDetail column are documented in the QualityMap table in the Runtime database.

```
SELECT DateTime, TagName, Value, QualityDetail FROM History
   WHERE TagName = 'SysTimeSec'
      AND DateTime >= '2005-04-11 11:19:10'
      AND DateTime <= '2005-04-11 11:21:10'
      AND wwRetrievalMode = 'Maximum'
      AND wwResolution = 60000
```

The results are:

	DateTime	TagName	Value	QualityDetail

	DateTime	TagName	Value	QualityDetail
(phantom cycle)	2005-04-11 11:19:10.000	SysTimeSec	NULL	65536
(cycle 1)	2005-04-11 11:19:59.000	SysTimeSec	59	4288
(cycle 2)	2005-04-11 11:20:59.000	SysTimeSec	59	192
(cycle 3)	2005-04-11 11:21:10.000	SysTimeSec	10	4288

Maximum Retrieval - Initial and Final Values

For analog tags, the maximum value of the tag in the cycle leading up to the query start time is returned with its timestamp changed to the query start time. If there is no point exactly at the phantom cycle start time, the point leading up to the phantom cycle is also considered for the maximum calculation. No adjustments are made to the quality of the initial point even though the timestamp may have been altered. Apart from the initial value, all points returned are delta points. (For more information on initial values, see *Determining Cycle Boundaries* on page 142.)

If a point occurs exactly on the query end time, that point is returned with the partial cycle bit, 4096, set in quality detail. If there is more than one such point, only the first point is returned.

Maximum Retrieval - Handling NULL Values and Incomplete Cycles

The first NULL value in a cycle is returned.

When a maximum value is returned from a cycle that contains gaps (including a gap extended from the previous cycle) or from an incomplete cycle with the query end time located inside of the calculation cycle, the point's quality detail is modified to flag this. This is done by performing a logical OR operation of the value 4096, which indicates a partial cycle, onto the existing quality detail.

As an example of how maximum retrieval mode handles NULLs, consider the following query:

```
SELECT TagName, DateTime, Value, QualityDetail
   FROM History
      WHERE TagName = 'A001'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwResolution = 10000
         AND wwRetrievalMode = 'Maximum'
```

If you run this query against the following sample data:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:09	0.2	192
A001	2009-09-12 00:15	1.3	192
A001	2009-09-12 00:17	0.8	192
A001	2009-09-12 00:22	0.5	192

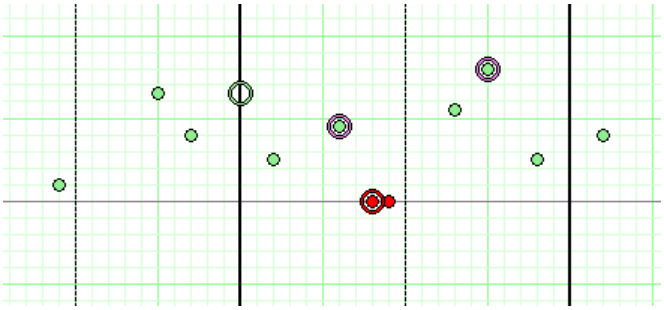
Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:26	0.9	192
A001	2009-09-12 00:28	0.0	249
A001	2009-09-12 00:29	0.0	249
A001	2009-09-12 00:33	1.1	192
A001	2009-09-12 00:35	1.6	192
A001	2009-09-12 00:38	0.5	192
A001	2009-09-12 00:42	0.8	192

The results are:

Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:20	1.3	192
A001	2009-09-12 00:26	0.9	4288
A001	2009-09-12 00:28	NULL	249
A001	2009-09-12 00:35	1.6	4288

The sample data points and the results are mapped on the following chart. Only the data falling between the time start and end marks at 00:20 and 00:40 (shown on the chart as dark vertical lines) are returned by the query. The resolution is set at 10,000 milliseconds.

Because there is no value that matches the start time, an initial value at 00:20 is returned based on the maximum value of the preceding cycle, which is the data point at 00:15. In the two subsequent cycles, the maximum values are at 00:26 and 00:35. The quality for these two values is set to 4288 (4096 + 192). The remaining data points are excluded because they are not maximums. In addition, the first NULL at 00:28 is included, but the second NULL (at 00:29) is not.



Integral Retrieval

Integral retrieval calculates the values at retrieval cycle boundaries by integrating the graph described by the points stored for the tag. Therefore, it works much like average retrieval, but it additionally applies a scaling factor. This retrieval mode is useful for calculating volume for a particular tag. For example, if one of your tags represents product flow in gallons per second, integral retrieval allows you to retrieve the total product flow in gallons during a certain time period.

Integral retrieval is a true cyclic mode. It returns one row for each tag in the query for each cycle. The number of cycles is based on the specified resolution or cycle count.

Integral retrieval only works with analog tags. For all other tags, normal cyclic results are returned.

Integral Retrieval - How It Works

Calculating values for a cycle in integral retrieval is a two-step process:

- First, the historian calculates the area under the graph created by the data points. This works the same as in average retrieval. For more information, see *Average Retrieval* on page 46.
- After this area has been found, it is scaled using the value of the IntegralDivisor column in the EngineeringUnit table. This divisor expresses the conversion factor from the actual rate to one of units per second.

For example, if the time-weighted average for a tag during a 1-minute cycle is 3.5 liters per second, integral retrieval returns a value of 210 for that cycle (3.5 liters per second multiplied by 60 seconds).

Integral Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in integral retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *Interpolation Type (wwInterpolationType)* on page 105
- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111

Integral Retrieval - Query Example

To use the integral retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'Integral'
```

In this example, the integral is computed for each of five 1-minute long cycles. The wwQualityRule parameter is used to ensure that only points with good quality are used in the computation, which means that points with doubtful quality are discarded. The rules used to determine the returned OPCQuality are the same as described for a time weighted average query.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 2), Value) AS Flow, OPCQuality,
PercentGood FROM History
   WHERE TagName = 'FlowRate'
      AND DateTime >= '2004-06-07 08:00'
      AND DateTime <  '2004-06-07 08:05'
      AND wwRetrievalMode = 'Integral'
      AND wwCycleCount = 5
      AND wwQualityRule = 'Good'
```

The results are:

	DateTime	TagName	Flow	OPCQuality	PercentGood

	DateTime	TagName	Flow	OPCQuality	PercentGood
(interval 1)	2004-06-07 08:00:00.000	FlowRate	2862.97	192	100.0
(interval 2)	2004-06-07 08:01:00.000	FlowRate	9436.85	192	100.0
(interval 3)	2004-06-07 08:02:00.000	FlowRate	2480.24	192	100.0
(interval 4)	2004-06-07 08:03:00.000	FlowRate	7379.71	192	100.0
(interval 5)	2004-06-07 08:04:00.000	FlowRate	6317.32	192	100.0

Also, the "phantom" cycle affects the integral retrieval mode just as it does the average retrieval mode. For examples, see *Querying Aggregate Data in Different Ways* on page 171.

Integral Retrieval - wwExpression Query Example

In this example, the integral of a tag named Speed (measured in m/s), is computed over two 1-minute long intervals (wwResolution = 60000 milliseconds) to measure the distance traveled in meters over the interval.

```
SELECT StartDateTime, DateTime, TagName, Value, Quality, QualityDetail as QD,
wwUnit
    FROM History
    WHERE TagName = 'Speed'
        AND DateTime >= '2020-12-03 10:00'
        AND DateTime <  '2020-12-03 10:02'
        AND wwRetrievalMode = 'Integral'
        AND wwResolution = 60000
```

The results are:

StartDateTime	DateTime	TagName	Value	Quality	QD	wwUnit
2020-12-03 09:59:00.000	2020-12-03 10:00:00.000	Speed	-2906.88948	0	192	m
2020-12-03 10:00:00.000	2020-12-03 10:01:00.000	Speed	877.57619	0	192	m

To display the results using a different unit of measure, you can use the wwExpression query parameter with the following syntax:

```
wwExpression = 'UOM(TagName, target unit)'
```

For example, the integral of the Speed tag is computed again over two 1-minute long intervals, but this time the value is converted to the distance in centimeters.

```
SELECT StartDateTime, DateTime, TagName, Value, Quality, QualityDetail as QD,
wwUnit
    FROM History
    WHERE TagName = 'Speed'
        AND DateTime >= '2020-12-03 10:00'
        AND DateTime <  '2020-12-03 10:02'
        AND wwRetrievalMode = 'Integral'
```

```
AND wwResolution = 60000
AND wwExpression = 'UOM([Speed],cm)'
```

The results are:

StartDateTime	DateTime	TagName	Value	Quality	QD	wwUnit
2020-12-03 09:59:00.000	2020-12-03 10:00:00.000	Speed	-290688.948	192	192	cm
2020-12-03 10:00:00.000	2020-12-03 10:01:00.000	Speed	87757.619	192	192	cm

Note: The engineering unit conversion feature is licensed separately. If you do not have the applicable license, the results are returned in the original units.

Integral Retrieval - Initial and Final Values

If wwTimeStampRule = END, the initial value is calculated by performing an integral calculation on the cycle leading up to the query start time. No special handling is done for the final value.

If wwTimeStampRule = START, the final value is calculated by performing an integral calculation on the cycle following the query end time. No special handling is done for the initial value.

Integral Retrieval - Handling NULL Values

Gaps introduced by NULL values are not included in the integral calculations. The average only considers the time ranges with good values. TimeGood indicates the total time per cycle that the tags value was good.

Slope Retrieval

Slope retrieval returns the slope of a line drawn through a given point and the point immediately before it, thus expressing the rate at which values change.

This retrieval mode is useful for detecting if a tag is changing at too great a rate. For example, you might have a temperature that should steadily rise and fall by a small amount, and a sharp increase or decrease could point to a potential problem.

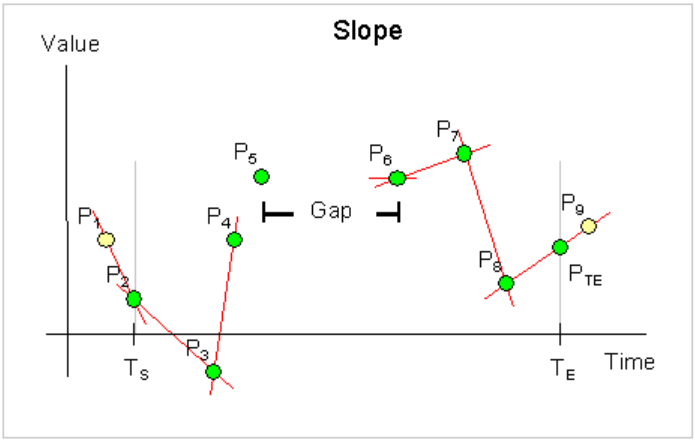
The slope retrieval mode can be considered a delta mode. Apart from the initial value and a value at the query end time, all returned points are calculated delta points returned with the timestamp of an actual delta point.

Slope retrieval only works with analog tags. For all other tags, normal delta results are returned.

All returned values are in chronological order. If multiple points are to be returned for a particular timestamp, they are returned in the order in which the tags were specified in the query.

Slope Retrieval - How It Works

The following illustration shows how the slope is calculated for an analog tag.



This example has a start time of T_S and an end time of T_E.

For the queried tag, a total of nine points are found, represented by the markers P_1 through P_9. Of these points, eight represent normal analog values. The point P_5 represents a NULL due to an I/O Server disconnect, which causes a gap in the data between P_5 and P_6.

For every point in the time period, slope retrieval returns the slope of the line going through that point and the point immediately before it. For two points P_1 and P_2 occurring at times T_1 and T_2, the slope formula is as follows:

$(P_2 - P_1) / (T_2 - T_1)$

The difference between T_1 and T_2 is measured in seconds. Therefore, the returned value represents the change in Engineering Units per second.

In this example, point P_2 is located at the query start time, and because there is a prior value (P_1), the slope of the line through both points is calculated and returned at time T_S. Similarly, slopes are calculated to be returned at times T_3, T_4, T_7, and T_8. The slope is also calculated for the line through P_8 and P_9, but that value is returned as point P_{TE} at the query end time.

For point P_6, there is no prior point with which to perform a slope calculation. Instead, the slope of the flat line going through the point (that is, the value 0) is calculated. At the time of P_5, NULL is returned.

The quality detail and OPC quality returned with a slope point is always directly inherited from the point that also provides the time stamp. In this example, this means that point P_2 provides the qualities for the slope point returned at the query start time, T_S.

Slope Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in slope retrieval mode. For more information, see the following sections:

- *History Version (wwVersion)* on page 104
- *Quality Rule (wwQualityRule)* on page 111

Slope Retrieval - Query Example

To use the slope retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'Slope'
```

For example, the following query calculates and returns the rate of change of the ReactTemp tag in °C/second. The initial value in the Quality column at the query start time shows no value is located exactly at that time, so the slope returned is the same as the one returned at the next delta point. (For more information on initial values, see *Determining Cycle Boundaries* on page 142.)

At 08:01:17.947 the tag has two delta points, so a slope is calculated and returned for the first point, while a NULL is returned at the second one with a special QualityDetail of 17, indicating that no slope can be calculated as it is either plus or minus infinite.

```
SELECT DateTime, TagName, CONVERT(DECIMAL(10, 4), Value) AS Slope, Quality,
QualityDetail FROM History
   WHERE TagName = 'ReactTemp'
         AND DateTime >= '2005-04-17 08:00'
         AND DateTime <= '2005-04-17 08:05'
         AND wwRetrievalMode = 'Slope'
```

The results are:

DateTime	TagName	Slope	Quality	QualityDetail
2005-04-17 08:00:00.000	ReactTemp	3.8110	133	192
2005-04-17 08:00:00.510	ReactTemp	3.8110	0	192
2005-04-17 08:00:01.713	ReactTemp	4.1563	0	192
2005-04-17 08:00:02.917	ReactTemp	4.1563	0	192
2005-04-17 08:00:04.230	ReactTemp	3.8081	0	192
2005-04-17 08:00:05.433	ReactTemp	4.1563	0	192
…	…	…		
2005-04-17 08:01:16.743	ReactTemp	-1.7517	0	192
2005-04-17 08:01:17.947	ReactTemp	-27.0158	0	192
2005-04-17 08:01:17.947	ReactTemp	NULL	1	17
2005-04-17 08:01:19.260	ReactTemp	-1.7530	0	192
2005-04-17 08:01:20.463	ReactTemp	-1.9119	0	192
2005-04-17 08:01:21.667	ReactTemp	-1.9119	0	192
2005-04-17 08:01:22.977	ReactTemp	-1.7517	0	192
…	…	…		

Slope Retrieval - wwExpression Query Example

The following query calculates and returns the rate of change of the Distance tag. The results are returned in the tag's default unit of measure, which in this case is feet/second.

```
SELECT DateTime, TagName, Value, QualityDetail, wwUnit FROM History
   WHERE TagName = 'Distance'
         AND wwRetrievalMode = 'Slope'
```

The results are:

DateTime	TagName	Value	QualityDetail	wwUnit
2020-07-29 06:50:53.748	Distance	1	192	ft/s
2020-07-29 06:50:54.748	Distance	2	192	ft/s

To display the results using a different unit of measure, you can use the wwExpression query parameter with the following syntax:

```
wwExpression = 'UOM(TagName, target unit)'
```

For example, the following query calculates and returns the rate of change of the Distance tag in meters/second.

```
SELECT DateTime, TagName, Value, QualityDetail, wwUnit FROM History
   WHERE TagName = 'Distance'
         AND wwExpression = 'UOM(Distance, m/s)'
         AND wwRetrievalMode = 'Slope'
```

The results are:

DateTime	TagName	Value	QualityDetail	wwUnit
2020-07-29 06:50:53.748	Distance	0.3048	192	m/s
2020-07-29 06:50:54.748	Distance	0.6096	192	m/s

Note: The engineering unit conversion feature is licensed separately. If you do not have the applicable license, the results are returned in the original units.

Slope Retrieval - Initial and Final Values

An initial value is always generated. If a point is stored exactly at the query start time, the slope is returned as the slope between that point and the previous point. Otherwise, the slope is calculated using the slope of the points before and after the query start time.

A final value is always generated. If a point is stored exactly at the query end time, the slope is returned as the slope between that point and the previous point. Otherwise, the slope is calculated using the slope of the points before and after the query end time.

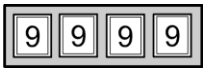
Slope Retrieval - Handling NULL Values

The first NULL following a non-NULL value is returned. Subsequent NULL values are not. If a point is preceded by a NULL, the slope for that point will be zero.

Counter Retrieval

Counter retrieval allows you to accurately retrieve the delta change of a tag's value over a period of time even for tags that are reset upon reaching a "rollover value." The rollover value is defined in the AVEVA Historian for each tag.

This retrieval mode is useful for determining how much of an item was produced during a particular time period. For example, you might have an integer counter that keeps track of how many cartons were produced. The counter has an indicator like this:



The next value after the highest value that can be physically shown by the counter is called the rollover value. In this example, the rollover value is 10,000. When the counter reaches the 9,999th value, the counter rolls back to 0. Therefore, a counter value of 9,900 at one time and a value of 100 at a later time means that you have produced 200 units during that period, even though the counter value has dropped by 9,800 (9,900 minus 100). Counter retrieval allows you to handle this situation and receive the correct value. For each cycle, the counter retrieval mode shows the increase in that counter during the cycle, including rollovers.

Note: If Historian receives a data tag value that is outside of the specified engineering unit boundaries, Historian ignores the received value and instead returns a 0. For example, if a value can be an integer between 1 and 8, and the received value is 9, Historian returns 0 as the value for that tag during that cycle.

Counter retrieval also works with floating point counters, which is useful for flow meter data. Similar to the carton counter, some flow meters "roll over" after a certain amount of flow accumulates. For both examples, the need is to convert the accumulating measure to a "delta change" value over a given period.

Counter retrieval is a true cyclic mode. It returns one row for each tag in the query for each cycle. The number of cycles is based on the specified resolution or cycle count.

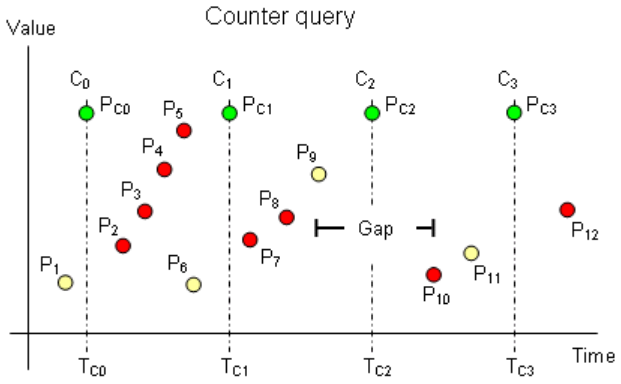
The counter algorithm is only applied to analog tags and to discrete tags. For integer analog tags, the result will be an integer returned as a float data type. For a real analog tag, the rollover value and the result may be real values and can include fractional values. If a query contains tags of other types, then no rows are returned for those tags. For discrete tags, the rollover value is assumed to be 2.

The rules used to determine the OPCQuality returned with a counter value are the same as for a time weighted average query. For more information, see *Quality Rule (wwQualityRule)* on page 111. When a rollover has occurred in the calculation cycle, a special quality detail of 212 is returned in all non-NULL cases.

CTU counters will default to "signed integer" tags when imported into the Historian, giving a normal range of -2147483648 to 2147483647 (for a 32-bit integer). In operation, these counters will count up to the upper limit and "rollover" to the lower limit on the next increment. If these tags are changed to be "unsigned integers" the normal range will be 0 to 4294967295 and values will rollover to "0", conforming to the expected behavior of a tag used with "counter" retrieval. When used with a 16-bit CTU counter, the same rules apply, but the range of values is -32768 to 32767 as "signed" and 0 to 65535 for an "unsigned".

Counter Retrieval - How It Works

The following illustration shows how the counter algorithm determines the count for an analog tag.



This example has a start time of T_{C0} and an end time of T_{C3}. The resolution has been set in such a way that the historian returns data for three complete cycles starting at T_{C0}, T_{C1}, and T_{C2}, and an incomplete cycle starting at T_{C3}.

For the queried tag, a total of twelve points are found throughout the cycles represented by the markers P_1 through P_{12}. Of these points, eleven represent normal analog values. The point P_9 represents a NULL due to an I/O Server disconnect, which causes a gap in the data between P_9 and P_{10}. Point P_{12} is not considered because it is outside of the query time frame.

All points are considered in the counter calculation, but only the yellow ones are actually used to determine which values to return to the client. The returned points are P_{C0}, P_{C1}, P_{C2} and P_{C3}, shown in green at the top to indicate that there is no simple relationship between them and any of the actual points.

All cycle values are calculated as the delta change between the cycle time in question and the previous cycle time, taking into account the number of rollovers between the two points in time. The counter algorithm assumes that a rollover occurred if the current value is lower than the previous value. The initial value at the query start time (P_{C1}) is calculated the same way, only based on a phantom cycle before the query start time.

For example, the formula to calculate P_{C1} is as follows:

$$P_{C1} = n * V_R + P_6 - P_1$$

where:

- n = the number of rollovers that have occurred during the cycle
- V_R = the rollover value for the tag

If either n or V_R are equal to zero, P_{C1} is simply the difference between the values P_1 and P_6.

In the case of cycle C_2, there is no value at the cycle time, so the NULL value represented by point P_9 is returned. In the case of cycle C_3, a NULL is again returned, because there is no counter value at the previous cycle boundary to use in the calculation. There must be a full cycle of values in order for the counter to be calculated.

If a gap is fully contained inside a cycle, and if points occur within the cycle on both sides of the gap, then a counter value is returned, even though it may occasionally be erroneous. Zero or one rollovers are assumed, even though the counter might have rolled over multiple times.

Counter Retrieval - Calculations for a Manually Reset Counter

If you have a counter that you typically reset manually before it rolls over, you must set the rollover value for the tag to 0 so that the count is simply how much change occurred since the manual reset.

For example, assume that you have the following data values for five consecutive cycle boundaries, and that the value 0 occurs as the first value within the last cycle:

100, 110, 117, 123, 3

If you set the rollover value to 0, the counter retrieval mode assumes that the 0 following the value 123 represents a manual reset, and returns a value of 3 for the last cycle, which is assumed to be the count after the manual reset. The value 0 itself does not contribute 1 to the counter value in this case.

If the rollover value is instead set to 200, then the counter retrieval mode assumes that the value 0 represents a normal rollover, and a count of 80 is calculated and returned (200 - 123 + 3). In this case, the value 0 contributes 1 to the counter value, and that is the change from the value 199 to the value 200.

Counter Retrieval - Using a Counter Deadband

You can set a deadband for counter retrieval to better handle reporting of rates and quantities. For example, setting a counter deadband can help to:

- Distinguish between counter resets and rollovers.
- Filter out counter reversals.

The counter deadband is the percentage of the full range of the counter.

For example, if you set the counter deadband to be 10, then any counter reset that occurs within the top 10% of the range is assumed to be a rollover and is counted as such.

For a reversal, you might have a conveyor belt carrying boxes that are counted as they pass by a station. If a jam occurs, you might reverse the conveyor belt to clear it, resulting in the counter decrementing and then incrementing again. In this case, a counter deadband of 10 would discard any duplicate counts within 10% of the range starting from the point of reversal. Only one reversal can be detected until the rollover occurs.

Counter Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in integral retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111

Counter Retrieval - Initial and Final Values

An initial value is returned using the period leading up to the query start time.

A data point that has a cycle time is used to generate the counter value for its preceding cycle. A NULL point with cycle time will cause the preceding cycle to end in a gap and the following cycle to start with a gap.

Counter Retrieval - Handling NULL Values

If wwQualityRule is configured as OPTIMISTIC, NULL data points will not be used in calculation. 0.0 will be used as the starting base value for the query unless the query data starts with a NULL. If the query starts with a NULL, the value change for the cycle is calculated from the first actual value in the cycle, rather than 0.

Otherwise, if any points considered in a cycle have UNCERTAIN quality, the result for that row will also have UNCERTAIN quality. Any cycle that starts or ends in a gap will have a quality detail of 65536.

The quality detail of DOUBTFUL will be used with the counter result for the cycles, if a NULL point is considered for the cycle and the counter result is not NULL.

Counter Retrieval - Handling Illegal Values

If the configured rollover value is larger than 0.0, then the data points whose values are greater than or equal to the rollover value causes the counter value for the cycle to be set to 0.0, with qdIO_FILTEREDPOINT applied to the quality detail.

Similarly, if any data point with a value less than 0.0 is found in a cycle, the counter value for the cycle is set to 0.0, with qdIO_FILTEREDPOINT applied to the quality detail.

Counter Retrieval - Query Example

To use the counter mode, set the following parameter in your query.

```
wwRetrievalMode = 'Counter'
```

In the following example, the rollover value for the SysTimeSec system tag is set to 0. In a five-minute time span, the SysTimeSec tag increments a total of 300 times. The following query returns the total count within the five-minute time span, which begins with StartTime and ends with DateTime. The QualityDetail of 212 indicates that a counter rollover occurred during the query time range.

```
select StartDateTime, DateTime, TagName, Value, wwUnit, QualityDetail as QD from
History
    where TagName = 'SysTimeSec'
        and DateTime > '2020-12-02 1:00'
        and DateTime <= '2020-12-02 1:05'
        and wwRetrievalMode = 'counter'
        and wwCycleCount = 1
```

The results are:

StartTime	DateTime	TagName	Value	wwUnit	QD
2020-12-02 01:00:00.0000000	2020-12-02 01:05:00.0000000	SysTimeSec	300	Second	212

To display the results using a different unit of measure, you can use the wwExpression query parameter with the following syntax:

```
    wwExpression = 'UOM(TagName, target unit)'
```

The following example is similar to the previous example. The following query returns the total count within the five-minute time span, which begins with StartTime and ends with DateTime, and converts the retrieved value from its original units (seconds) to minutes.

```
select StartDateTime, DateTime, TagName, Value, Quality, QualityDetail as QD
from History
    where TagName = 'SysTimeSec'
        and DateTime > '2020-12-02 1:00'
        and DateTime <= '2020-12-02 1:05'
        and wwRetrievalMode = 'counter'
        and wwCycleCount = 1
        and wwExpression = 'UOM(SysTimeSec,Minute)'
```

The results are:

StartTime	DateTime	TagName	Value	wwUnit	QD
2020-12-02 01:00:00.0000000	2020-12-02 01:05:00.0000000	SysTimeSec	5	Minute	212

Note: The engineering unit conversion feature is licensed separately. If you do not have the applicable license, the results are returned in the original units.

ValueState Retrieval

ValueState retrieval returns information on how long a tag has been in a particular value state during each retrieval cycle. That is, a time-in-state calculation is applied to the tag value.

This retrieval mode is useful for determining how long a machine has been running or stopped, how much time a process spent in a particular state, how long a valve has been open or closed, and so on. For example, you might have a steam valve that releases steam into a reactor, and you want to know the average amount of time the valve was in the open position during the last hour. ValueState retrieval can return the shortest, longest, average, or total time a tag spent in a state, or the time spent in a state as a percentage of the total cycle length.

When you use ValueState retrieval for a tag in a trend chart, you must specify a single value state for which to retrieve information. ValueState retrieval then returns one value for each cycle—for example, the total amount of time that the valve was in the "open" state during each 1-hour cycle. This information is suitable for trend display.

If you *don't* specify a state, ValueState retrieval returns one row of information for each value that the tag was in during each cycle. For example, this would return not only the time a valve was in the "open" state, but also the time it was in the "closed" state. This information is not suitable for meaningful display in a regular trend. You can, however, retrieve this type of information in a query and view it as a table.

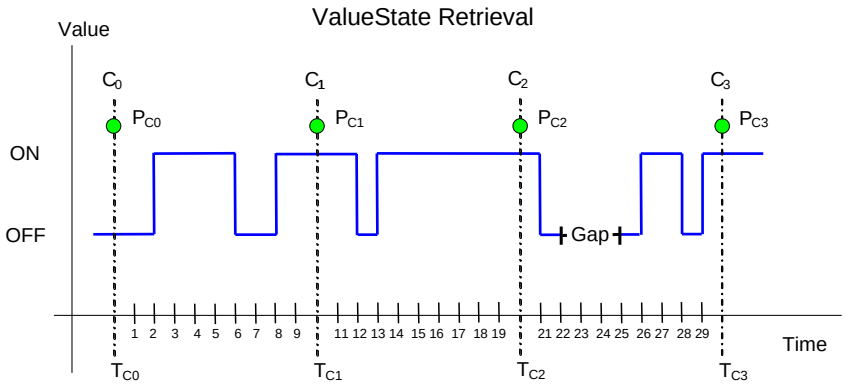
ValueState retrieval works with integer, discrete, string, and state summary tags. For other types of tags, no rows are returned. NULL values are treated like any other distinct state.

The values returned at the query start time are the result of applying the algorithm to a "phantom" cycle preceding the query range. It is assumed that the tag value at the start of the cycle is located at that point in time.

To specify the type of calculation, set the wwStateCalc parameter in the query. For more information, see *State Calculation (wwStateCalc)* on page 119.

ValueState Retrieval - How It Works

The following illustration shows how ValueState retrieval returns values for a discrete tag.



This example has a start time of T_{C0} and an end time of T_{C3}. The resolution has been set in such a way that the historian returns data for three complete cycles starting at T_{C0}, T_{C1}, and T_{C2}, and an incomplete cycle starting at T_{C3}. Time is measured seconds.

A gap in the data occurs in the third cycle due to an I/O Server disconnect.

The state calculation is based on each cycle, and the values returned at the query start time are not regular initial values, but are the resulting values that occur after applying the algorithm to the last cycle preceding the query range. The returned points are P_{C0}, P_{C1}, P_{C2} and P_{C3}, shown in green at the top to indicate that there is no simple relationship between the calculated values and any of the actual points.

Assume the query is set so that the total time (wwStateCalc = 'Total')in the two states are returned. The timestamping is set to use the cycle end time.

- For T_{C0}, the query returns two rows (one for the "on" state and one for the "off" state), calculated as a result of the "phantom" cycle that precedes the query start time. The values have a timestamp of the query start time.
- For T_{C1}, one row is returned for the "on" state. The "on" state occurred twice during the cycle--one time for four seconds and again for two seconds before the cycle boundary, and the total time returned is six seconds. The state was "off" twice during the cycle for a total time of four seconds, and one row is returned with that value.
- For T_{C2}, one row is returned for the "on" state, and one row is returned for the "off" state. The "on" state occurred for a total of nine seconds between the cycle boundaries, and the "off" state occurred for a total of one second.
- For T_{C3}, one row is returned for the "on" state, and one row is returned for the "off" state. The "on" state occurred for a total of four seconds between the cycle boundaries, and the "off" state occurred for a total of three seconds. An additional row is returned for the NULL state occurring as a result of the I/O Server disconnect.

Using the same data, if you queried the **total contained** time for the states, the following is returned:

- For T_{C0}, the query returns two values (one for the "on" state and one for the "off" state), calculated as a result of the "phantom" cycle the precedes the query start time. The value has a timestamp of the query start time.
- For T_{C1}, one row is returned for the "on" state, and one row is returned for the "off" state. The "on" state occurred one time for four seconds within the cycle. The two seconds of "on" time that crosses the cycle boundary does not contribute to the total time. The state was "off" one time during the cycle for two seconds completely within the cycle boundary.
- For T_{C2}, the state was not "on" for any contained time between the cycle. Both occurrences of the "on" state cross over a cycle boundary, so no rows are returned for this state. One row is returned for the "off" state. The state was "off" one time during the cycle for one seconds completely within the cycle boundary.
- For T_{C3}, one row is returned for the "on" state, and one row is returned for the "off" state. The state was "on" for a single contained time of two seconds between the cycle boundaries. The state was "off" three times during the cycle for three seconds completely within the cycle boundary. An additional row is returned for the NULL state occurring as a result of the I/O Server disconnect. The state was NULL for a total of three seconds. The I/O Server disconnect that "disrupted" the off state is treated as its own state, thereby changing what would have been a single "off" state instance of five seconds into two instances of the "off" state for one second each.

ValueState Retrieval - Supported Value Parameters

You can use various parameters to adjust which values are returned in ValueState retrieval mode. For more information, see the following sections:

- *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91
- *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93
- *History Version (wwVersion)* on page 104
- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111
- *State Calculation (wwStateCalc)* on page 119

ValueState Retrieval - Query Examples

To use the ValueState retrieval mode, set the following parameter in your query.

```
wwRetrievalMode = 'ValueState'
```

To specify the type of aggregation, set the wwStateCalc parameter in the query, such as:

```
wwStateCalc = 'Total'
```

Be sure that you use the "<=" operator for ending date/time.

For examples, see the following:

ValueState Retrieval Query 1: Minimum Time in State

The following query finds the minimum time-in-state for the SteamValve discrete tag. Note that minimum times are returned for each state for both the five-minute phantom cycle before the query start time and for the single retrieval cycle between 10:00 and 10:05.

```
SELECT DateTime, TagName, vValue, StateTime, wwStateCalc FROM History
   WHERE TagName IN ('SteamValve')
      AND DateTime >= '2005-04-17 10:00:00'
      AND DateTime <= '2005-04-17 10:05:00'
      AND wwCycleCount = 1
      AND wwRetrievalMode = 'ValueState'
      AND wwStateCalc = 'Min'
```

The results are:

DateTime	TagName	vValue	StateTime	wwStateCalc
2005-04-17 10:00:00.000	SteamValve	0	35359.0	MINIMUM
2005-04-17 10:00:00.000	SteamValve	1	43749.0	MINIMUM
2005-04-17 10:05:00.000	SteamValve	0	37887.0	MINIMUM
2005-04-17 10:05:00.000	SteamValve	1	43749.0	MINIMUM

ValueState Retrieval Query 2: Minimum Time in State for a Single Tag

The following query finds the minimum time-in-state for the Mixer discrete tag for the "on" state. Note that minimum times are returned for each state for both the five-minute phantom cycle before the query start time and for the single retrieval cycle between 10:00 and 10:05.

```
SELECT DateTime, TagName, vValue, StateTime, wwStateCalc
   FROM History
      WHERE TagName IN ('Mixer')
         AND DateTime >= '2017-12-11 08:00:00'
```

```
AND DateTime < '2017-12-12 08:00:00'
AND wwCycleCount= '1'
AND wwRetrievalMode = 'ValueState'
AND wwStateCalc = 'Min'
AND vValue = 1
```

The results are:

DateTime	TagName	vValue	StateTime	wwStateCalc
2017-12-11 08:00:00.0000000	Mixer	1	35906	Min

ValueState Retrieval Query 3

The following query finds the maximum time-in-state for the SteamValve discrete tag in the same time period as in Query 1. Note how both the minimum and maximum values for the "1" state are very similar, while they are very different for the "0" state. This is due to the "cut-off" effect.

```
SELECT DateTime, TagName, vValue, StateTime, wwStateCalc FROM History
   WHERE TagName IN ('SteamValve')
      AND DateTime >= '2005-04-17 10:00:00'
      AND DateTime <= '2005-04-17 10:05:00'
      AND wwCycleCount = 1
      AND wwRetrievalMode = 'ValueState'
      AND wwStateCalc = 'Max'
```

The results are:

DateTime	TagName	vValue	StateTime	wwStateCalc
2005-04-17 10:00:00.000	SteamValve	0	107514.0	MAXIMUM
2005-04-17 10:00:00.000	SteamValve	1	43750.0	MAXIMUM
2005-04-17 10:05:00.000	SteamValve	0	107514.0	MAXIMUM
2005-04-17 10:05:00.000	SteamValve	1	43750.0	MAXIMUM

ValueState Retrieval Query 4

The following query returns the total of time in state for a discrete tag. In this example, the TimeStampRule system parameter is set to "End" (the default setting), so the returned values are timestamped at the end of the cycle. The returned rows represent the time-in-state behavior during the period starting at 2005-04-13 00:00:00.000 and ending at 2005-04-14 00:00:00.000.

```
SELECT DateTime, vValue, StateTime, wwStateCalc FROM History
   WHERE DateTime > '2005-04-13 00:00:00.000'
      AND DateTime <= '2005-04-14 00:00:00.000'
      AND TagName IN ('PumpStatus')
      AND wwRetrievalMode = 'ValueState'
      AND wwStateCalc = 'Total'
      AND wwCycleCount = 1
```

The results are:

DateTime	vValue	StateTime	wwStateCalc
2005-04-14 00:00:00	NULL	1041674.0	TOTAL
2005-04-14 00:00:00	On	56337454.0	TOTAL

DateTime	vValue	StateTime	wwStateCalc
2005-04-14 00:00:00	Off	29020872.0	TOTAL

ValueState Retrieval Query 5

The following query returns the percentage of time in state for a discrete tag for multiple retrieval cycles. The TimeStampRule system parameter is set to "End" (the default setting), so the returned values are timestamped at the end of the cycle. Note that the first row returned represents the results for the period starting at 2003-07-03 22:00:00.000 and ending at 2003-07-04 00:00:00.000.

The "Percent" time-in-state retrieval mode is the only mode in which the StateTime column does not return time data. Instead, it returns percentage data (in the range of 0 to 100 percent) representing the percentage of time in state.

```
SELECT DateTime, vValue, StateTime, wwStateCalc FROM History
   WHERE DateTime >= '2003-07-04 00:00:00.000'
      AND DateTime <= '2003-07-05 00:00:00.000'
      AND TagName IN ('PumpStatus')
      AND Value = 1
      AND wwRetrievalMode = 'ValueState'
      AND wwStateCalc = 'Percent'
      AND wwCycleCount = 13
```

The results are:

DateTime	vValue	StateTime	wwStateCalc
2003-07-04 00:00:00	1	50.885	PERCENT
2003-07-04 02:00:00	1	82.656	PERCENT
2003-07-04 04:00:00	1	7.082	PERCENT
2003-07-04 06:00:00	1	7.157	PERCENT
2003-07-04 08:00:00	1	55.580	PERCENT
2003-07-04 10:00:00	1	28.047	PERCENT
2003-07-04 12:00:00	1	47.562	PERCENT
2003-07-04 14:00:00	1	74.477	PERCENT
2003-07-04 16:00:00	1	40.450	PERCENT
2003-07-04 18:00:00	1	78.313	PERCENT
2003-07-04 20:00:00	1	54.886	PERCENT
2003-07-04 22:00:00	1	39.569	PERCENT
2003-07-05 00:00:00	1	50.072	PERCENT

ValueState Retrieval Query 6: Querying State Summary Values

If state summary values are queried and the cycle boundaries match the summary periods, the ValueState calculations are supported and return valid results.

If state summary points are queried and the cycle boundaries do not match the summary periods, the ValueState calculations are supported, but they return DOUBTFUL (QualityDetail = 64) results.

State summaries are included in the cycle where the summary end time occurs. This causes results that do not match queries against the source tag and may cause inaccurate results, such as a total state time that is greater than the cycle time.

For example, this can occur if SysTimeSec is summarized with a state summary with one minute resolution, but then queried with 10 second intervals. In most of the retrieval cycles, there will be no values, but in the cycle that includes the summary end time (one in six of the retrieval cycles), all 60 states would be returned with each state having a state time of 1 second for a total of 60 seconds of state time in a 10 second retrieval cycle.

ValueState Retrieval - Initial and Final Values

The values returned at the query start time are the result of applying the algorithm to the last cycle preceding the query range.

ValueState Retrieval - Handling NULL Values

NULLs are considered a state and are reported along with the other states.

RoundTrip Retrieval

RoundTrip retrieval is very similar to ValueState retrieval in that it performs calculations on state occurrences in the within a cycle period you specify. However, ValueState retrieval uses the time spent in a certain state for the calculation, and RoundTrip retrieval uses the time between consecutive leading edges of the same state for its calculations.

You can use the RoundTrip retrieval mode for increasing the efficiency of a process. For example, if a process produces one item per cycle, then you would want to minimize the time lapse between two consecutive cycles.

The RoundTrip mode returns a row for each state in any given cycle. RoundTrip retrieval only works with integer analog tags, discrete tags, and string tags. If real analog tags are specified in the query, then no rows are returned for these tags. RoundTrip retrieval is not applied to state summary or analog summary tags. NULL values are treated as any other distinct value and are used to analyze the round trip for disturbances.

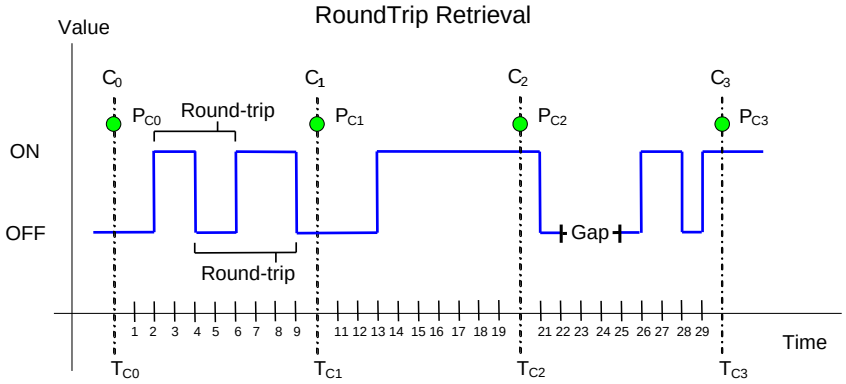
RoundTrip retrieval is supported for the History and StateWideHistory tables.

Any point on the boundary of the end cycle will be considered to the next cycle. The point on the boundary of the end query range is not counted in the calculation except that it is used to indicate that the previous state is a contained state.

If no roundtrip state is found within the cycle for a supported tag, a NULL StateTime value is returned. If there is no valid point prior to the phantom cycle, a NULL state is returned for the phantom cycle.

RoundTrip Retrieval - How It Works

The following illustration shows how RoundTrip retrieval returns values for a discrete tag.



This example has a start time of T_{C0} and an end time of T_{C3}. The resolution has been set in such a way that the historian returns data for three complete cycles starting at T_{C0}, T_{C1}, and T_{C2}, and an incomplete cycle starting at T_{C3}. Time is measured seconds.

A gap in the data occurs in the third cycle due to an I/O Server disconnect.

The state calculation is based on each cycle, and the values returned at the query start time are not regular initial values, but are the resulting values that occur after applying the algorithm to the last cycle preceding the query range. The returned points are P_{C0}, P_{C1}, P_{C2} and P_{C3}, shown in green at the top to indicate that there is no simple relationship between the calculated values and any of the actual points.

Assume the query is set so that the total contained time in the two states are returned. The timestamping is set to use the cycle end time. The RoundTrip retrieval mode returns values for states that are completely contained within the cycle boundaries. The following is returned:

- For T_{C0}, the query returns two values (one for the "on" state and one for the "off" state), calculated as a result of the "phantom" cycle that preceeds the query start time. The value has a timestamp of the query start time.
- For T_{C1}, one row is returned for the "on" state, and one row is returned for the "off" state. The round-trip for the "on" state occurred one time for four seconds completely within the cycle boundary. The round-trip for the "off" state occurred one time during the cycle for five seconds.
- For T_{C2}, a round-trip did not occur for either state within the cycle boundaries. One NULL row is returned for this cycle.
- For T_{C3}, one row is returned for the "on" state, and one row is returned for the "off" state. The state was "on" for a single contained time of two seconds between the cycle boundaries. The state was "off" one time during the cycle for one second completely within the cycle boundary. An additional row is returned for the NULL state occurring as a result of the I/O Server disconnect.
- For T_{C3}, one row is returned for the "on" state, and one row is returned for the "off" state. The state was "on" for a single contained time of three seconds between the cycle boundaries. One row is returned for the "off" state for a total contained time of seven seconds. (The first round-trip for the "off" state includes the I/O Server disconnect for a length of four seconds. The second round-trip has a length of three seconds.) An additional row is returned for the NULL state occurring as a result of the I/O Server disconnect, and the returned value is NULL because there is no round-trip during the cycle for it. The I/O Server disconnect that "disrupted" the off state is treated as its own state, thereby changing what would have been a single "off" state instance of five seconds into two instances of the "off" state for one second each.

RoundTrip Retrieval - Supported Value Parameters

You can use various parameters to adjust the values that must be returned in the RoundTrip retrieval mode. For more information, see the following sections:

- *TimeStamp Rule (wwTimeStampRule)* on page 107
- *Quality Rule (wwQualityRule)* on page 111
- *State Calculation (wwStateCalc)* on page 119

RoundTrip Retrieval - Query Examples

To use the RoundTrip retrieval mode, set the following parameter in your query:

```
wwRetrievalMode = 'RoundTrip'
```

The following queries compare the results between ValueState retrieval and RoundTrip retrieval.

This first ValueState retrieval query returns the average amount of time that the 'Reactor1OutletValve' tag is in "on" state and the average amount of time it is in the "off" state for a single cycle. Any state changes that occur across the cycle boundaries are not included.

```
SELECT DateTime, vValue, StateTime
   FROM History
      WHERE TagName IN ('Reactor1OutletValve')
      AND DateTime >= '2009-09-16 12:35:00'
      AND DateTime <= '2009-09-16 12:55:00'
      AND wwRetrievalMode = 'ValueState'
      AND wwStateCalc = 'AvgContained'
      AND wwCycleCount = 1
```

The results are:

DateTime	vValue	StateTime
2009-09-16 12:35:00.0000000	0	215878
2009-09-16 12:35:00.0000000	1	61729
2009-09-16 12:55:00.0000000	1	62827.5
2009-09-16 12:55:00.0000000	0	212856

The first two rows are for the "phantom" cycle leading up to the query start time and have a timestamp of the query start time.

The second two rows show the average amount of time for each state and have a timestamp of the query end time (the default).

Compare these results to same basic query that instead uses RoundTrip retrieval:

```
SELECT DateTime, vValue, StateTime
   FROM History
      WHERE TagName IN ('Reactor1OutletValve')
      AND DateTime >= '2009-09-16 12:35:00'
      AND DateTime <= '2009-09-16 12:55:00'
      AND wwRetrievalMode = 'RoundTrip'
      AND wwStateCalc = 'AvgContained'
      AND wwCycleCount = 1
```

The results are:

DateTime	vValue	StateTime

DateTime	vValue	StateTime
2009-09-16 12:35:00.0000000	1	277607
2009-09-16 12:35:00.0000000	0	278580
2009-09-16 12:55:00.0000000	0	275683.5
2009-09-16 12:55:00.0000000	1	273845

RoundTrip Retrieval - Initial and Final Values

The values returned at the query start time are the result of applying the algorithm to the last cycle preceding the query range.

RoundTrip Retrieval - Handling NULL Values

Like in the ValueState retrieval mode, the NULL state is treated as a valid distinct value. This allows you to analyze round trips for disturbances.

Edge Detection for Events (wwEdgeDetection)

For AVEVA Historian, an event is the moment at which a detection criterion is met on historical tag values in the AVEVA Historian. At a basic level, anything that can be determined by looking at stored data can be used as an event.

When detecting events, it is useful to pinpoint rows in a result set where the satisfaction of criteria in a WHERE clause changed from true to false, or vice versa. For example, you may want to know when the level of a tank went above 5 feet. The WHERE clause in a query for this example might be TANKLEVEL > 5. As the tank level approaches 5 feet, the criterion does not return true. Only when the level crosses the line from not satisfying the criterion to satisfying it, does the event actually occur. This imaginary "line" where the change occurs is called the edge.

Over a period of time, there may be many instances where the criteria cross the "edge" from being satisfied to not satisfied, and vice-versa. The values on either side of this "edge" can be detected if you configure your event tag to include this information. There are four possible options for edge detection: none, leading, trailing, or both. You will get differing results based on which option you use:

Option	Results
None	Returns all rows that successfully meet the criteria; no edge detection is implemented at the specified resolution.
Leading	Returns only rows that are the first to successfully meet the criteria (return true) after a row did not successfully meet the criteria (returned false).
Trailing	Returns only rows that are the first to fail the criteria (return false) after a row successfully met the criteria (returned true).
Both	All rows satisfying both the leading and trailing conditions are returned.

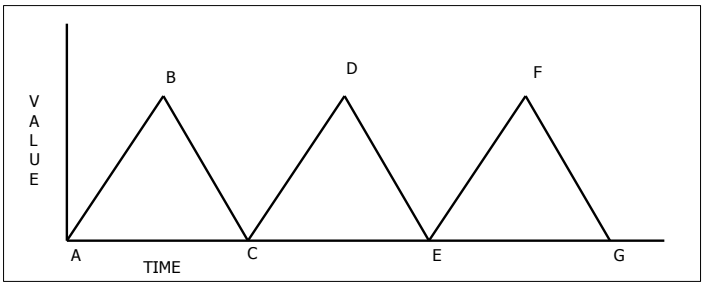
Edge detection only applies to analog and discrete value detectors. Also, edge detection is handled slightly differently based on whether you are using analog tags or discrete tags.

You can also use the ToDiscrete() query filter to determine when data values cross a particular threshold. For more information, see *Converting Analog Values to Discrete Values (ToDiscrete)* on page 122.

For more information on event detection with the Classic Event subsystem, see Classic Event Subsystem in the *AVEVA Historian Supplemental Guide*.

Edge Detection for Analog Tags

For example, the behavior of the WHERE clause as it processes a result set can be illustrated as:



Slopes A-B, C-D and E-F are rising edges, while slopes B-C, D-E and F-G are falling edges. The slopes are affected by the WHERE clause, which is a combination of the wwEdgeDetection option and the comparison operator used against the value.

The following table describes the rows that would be returned for the various edge detection settings:

Operator	**=**	**<**	**>**	**<=**	**>=**
Leading	Falling and rising edges; first value that meets the criteria.	Falling edge only; first value to meet the criteria.*	Rising edge only; first value to meet the criteria.	Falling edge only; first value to meet the criteria. *	Rising edge only; first value to meet the criteria.
Trailing	Falling and rising edges; first value to fail the criteria after a value meets the criteria.	Rising edge only; equal to the first value to fail the criteria.	Falling edge only; first value to fail the criteria.*	Rising edge only; first value to fail the criteria.	Falling edge only; first value to fail the criteria.*

* If the falling edge is a vertical edge with no slope, the query will return the lowest value of that edge.

The following query selects all values of "SysTimeSec" that are greater than or equal to 50 from the History table between 10:00 and 10:02 a.m. on December 2, 2001. No edge detection is specified.

```
SELECT DateTime, Value
   FROM History
```

```
    WHERE TagName = 'SysTimeSec'
        AND DateTime >= '2001-12-02 10:00:00'
        AND DateTime <= '2001-12-02 10:02:00'
        AND wwRetrievalMode = 'Cyclic'
        AND wwResolution = 2000
        AND Value >= 50
        AND wwEdgeDetection = 'None'
```

The results are:

DateTime	Value
2001-12-02 10:00:50.000	50
2001-12-02 10:00:52.000	52
2001-12-02 10:00:54.000	54
2001-12-02 10:00:56.000	56
2001-12-02 10:00:58.000	58
2001-12-02 10:01:50.000	50
2001-12-02 10:01:52.000	52
2001-12-02 10:01:54.000	54
2001-12-02 10:01:56.000	56
2001-12-02 10:01:58.000	58

Leading Edge Detection for Analog Tags

If Leading is specified as the parameter in the edge detection time domain extension, the only rows in the result set are those that are the first to successfully meet the WHERE clause criteria (returned true) after a row did not successfully meet the WHERE clause criteria (returned false).

The following query selects the first values of "SysTimeSec" from the History table to meet the Value criterion between 10:00 and 10:02 a.m. on December 2, 2001.

```
SELECT DateTime, Value
    FROM History
        WHERE TagName = 'SysTimeSec'
            AND DateTime >= '2001-12-02 10:00:00'
            AND DateTime <= '2001-12-02 10:02:00'
            AND wwRetrievalMode = 'Cyclic'
            AND wwResolution = 2000
            AND Value >= 50
            AND wwEdgeDetection = 'Leading'
```

The query will return only the two values that were greater than or equal to 50 for the time period specified:

DateTime	Value
2001-12-02 10:00:50.000	50
2001-12-02 10:01:50.000	50

Compare these results with the same query using no edge detection, as shown in *Edge Detection for Analog Tags* on page 79. Notice that even though the original query returned ten rows, the edge detection only returns the first row recorded after the event criteria returned true.

Trailing Edge Detection for Analog Tags

If Trailing is specified as the parameter in the edge detection extension, the only rows in the result set are those that are the first to fail the criteria in the WHERE clause (returned false) after a row successfully met the WHERE clause criteria (returned true).

The following query selects the first values of "SysTimeSec" from the History table to fail the Value criterion between 10:00 and 10:02 a.m. on December 2, 2001.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-02 10:00:00'
         AND DateTime <= '2001-12-02 10:02:00'
         AND wwRetrievalMode = 'Cyclic'
         AND wwResolution = 2000
         AND Value >= 50
         AND wwEdgeDetection = 'Trailing'
```

The query returns only the two values that were the first to fail the criteria in the WHERE clause for the time period specified:

DateTime	Value
2001-12-02 10:01:00.000	0
2001-12-02 10:02:00.000	0

Compare these results with the same query using no edge detection, as shown in *Edge Detection for Analog Tags* on page 79. Notice that even though the original query returned ten recorded rows for each value, the edge detection only returns the first row recorded after the event criteria returned false.

Both Leading and Trailing Edge Detection for Analog Tags

If Both is specified as the parameter in the edge detection extension, all rows satisfying both the leading and trailing conditions are returned.

The following query selects values of "SysTimeSec" from the History table that meet both the Leading and Trailing criteria between 10:00 and 10:02 a.m. on December 2, 2001.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-02 10:00:00'
         AND DateTime <= '2001-12-02 10:02:00'
         AND wwRetrievalMode = 'Cyclic'
         AND wwResolution = 2000
         AND Value >= 50
         AND wwEdgeDetection = 'Both'
```

The results are:

DateTime	Value
2001-12-02 10:00:50.000	50
2001-12-02 10:01:00.000	0
2001-12-02 10:01:50.000	50
2001-12-02 10:02:00.000	0

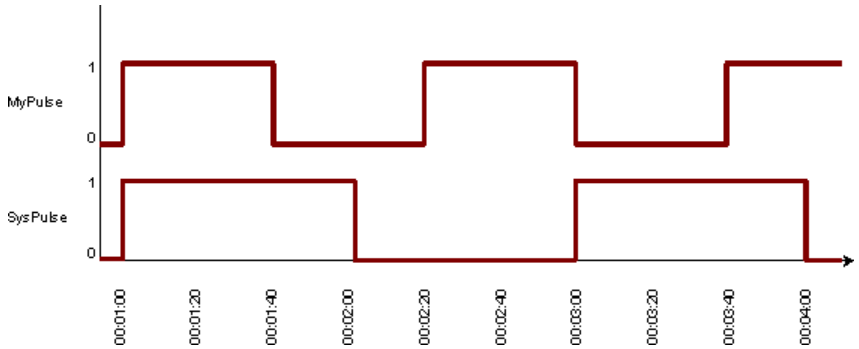
Compare these results with the same query using no edge detection, as shown in *Edge Detection for Analog Tags* on page 79. Notice that value of the first row in the original query is returned in the result set.

Edge Detection for Discrete Tags

Edge detection for discrete tags operates differently than for analog tags. For example, assume the following discrete tags are stored.

Tag	Description
SysPulse	Transitions between 1 and 0 every minute.
MyPulse	Transitions between 1 and 0 every 40 seconds.

A representation of the data stored in the system is as follows:



The following queries select values of "SysPulse" and "MyPulse" that have a value of 1 (On) from the History and WideHistory tables between 12:59 and 1:04 a.m. on December 8, 2001. No edge detection is specified.

This is a query of the History table:

```
SELECT DateTime, TagName, Value
    FROM History
        WHERE TagName IN ('SysPulse','MyPulse')
            AND DateTime > '2001-12-08 00:59:00'
            AND DateTime <= '2001-12-08 01:04:00'
            AND wwRetrievalMode = 'Delta'
            AND Value = 1
            AND wwEdgeDetection = 'None'
```

The results are:

DateTime	TagName	Value
2001-12-08 00:01:00.000	SysPulse	1
2001-12-08 00:01:00.000	MyPulse	1
2001-12-08 00:02:20.000	MyPulse	1
2001-12-08 00:03:00.000	SysPulse	1
2001-12-08 00:03:40.000	MyPulse	1

The following is a query of the WideHistory table:

```
SELECT * FROM OpenQuery(INSQL, 'SELECT DateTime, SysPulse, MyPulse FROM
WideHistory
   WHERE DateTime >  "2001-12-08 00:59:00"
      AND DateTime < "2001-12-08 01:05:00"
      AND SysPulse = 1
      AND MyPulse = 1
      AND wwRetrievalMode = "Delta"
      AND wwEdgeDetection = "None"
   ')
```

The results are:

DateTime	SysPulse	MyPulse
2001-12-08 00:01:00.000	1	1

Leading Edge Detection for Discrete Tags

If Leading is specified as the parameter in the edge detection time domain extension, the only rows in the result set are those that are the first to successfully meet the WHERE clause criteria (returned true) after a row did not successfully meet the WHERE clause criteria (returned false).

The following queries select values of "SysPulse" and "MyPulse" that have a value of 1 (On) from the History and WideHistory tables between 12:59 and 1:04 a.m. on December 8, 2001.

This example queries the History table, if the WHERE clause criteria specify to return only discrete values equal to 1 (On), then applying a leading edge detection does not change the result set.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysPulse','MyPulse')
         AND DateTime > '2001-12-08 00:59:00'
         AND DateTime <= '2001-12-08 01:04:00'
         AND Value = 1
         AND wwEdgeDetection = 'Leading'
```

The results are:

DateTime	TagName	Value
2001-12-08 00:01:00.000	SysPulse	1
2001-12-08 00:01:00.000	MyPulse	1
2001-12-08 00:02:20.000	MyPulse	1
2001-12-08 00:03:00.000	SysPulse	1
2001-12-08 00:03:40.000	MyPulse	1

This example queries the WideHistory table, applying a leading edge detection requires that the condition only evaluate to true if both values are equal to 1 (On).

```
SELECT DateTime, SysPulse, MyPulse FROM OpenQuery(INSQL, 'SELECT DateTime,
SysPulse, MyPulse
   FROM WideHistory
      WHERE DateTime >  "2001-12-08 00:59:00"
         AND DateTime <= "2001-12-08 01:04:00"
         AND SysPulse = 1
         AND MyPulse = 1
         AND wwEdgeDetection = "Leading"
      ')
```

The results are:

DateTime	SysPulse	MyPulse
2001-12-08 00:01:00.000	1	1
2001-12-08 00:03:40.000	1	1

Compare these results with the same query using no edge detection, as shown in *Edge Detection for Discrete Tags* on page 82. If you look at the diagram, you might think that a row could be returned at 00:03:00, but because both tags change at exactly this instant, their values are not returned. In a normal process, it is unlikely that two tags would change at exactly at the same instant.

Trailing Edge Detection for Discrete Tags

If Trailing is specified as the parameter in the edge detection extension, the only rows in the result set are those that are the first to fail the criteria in the WHERE clause (returned false) after a row successfully met the WHERE clause criteria (returned true).

This example queries the History table. If the WHERE clause criteria specifies returning only discrete values equal to 1 (On), then applying a trailing edge detection is the same as reversing the WHERE clause (as if Value = 0 was applied).

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysPulse','MyPulse')
         AND DateTime > '2001-12-08 00:59:00'
         AND DateTime <= '2001-12-08 01:04:00'
         AND Value = 1
```

```
        AND wwEdgeDetection = 'Trailing'
```

The results are:

DateTime	TagName	Value
2001-12-08 00:01:40.000	MyPulse	1
2001-12-08 00:02:00.000	SysPulse	1
2001-12-08 00:03:00.000	MyPulse	1
2001-12-08 00:04:00.000	SysPulse	1

This example queries the WideHistory table. It applies a trailing edge detection returns the boundaries where the condition ceases to be true (one of the values is equal to 0).

```
SELECT DateTime, SysPulse, MyPulse FROM OpenQuery(INSQL, 'SELECT DateTime,
SysPulse, MyPulse
   FROM WideHistory
      WHERE DateTime >  "2001-12-08 00:59:00"
         AND DateTime <= "2001-12-08 01:04:00"
         AND SysPulse = 1
         AND MyPulse = 1
         AND wwEdgeDetection = "Trailing"
      ')
```

The results are:

DateTime	SysPulse	MyPulse
2001-12-08 00:01:40.000	1	1
2001-12-08 00:04:00.000	1	1

Compare these results with the same query using no edge detection, as shown in *Edge Detection for Discrete Tags* on page 82. If you look at the diagram, you might think that a row could be returned at 00:03:00, but because both tags change at exactly this instant, their values are not returned. In a normal process, it is unlikely that two tags would change at exactly at the same instant.

Both Leading and Trailing Edge Detection for Discrete Tags

If Both is specified as the parameter in the edge detection extension, all rows satisfying both the leading and trailing conditions are returned.

The following queries select values of "SysPulse" and "MyPulse" that meet an edge detection of Both for a value criterion of 1 (On) from the History and WideHistory tables between 12:59 and 1:04 a.m. on December 8, 2001.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysPulse','MyPulse')
         AND DateTime > '2001-12-08 00:59:00'
```

```
AND DateTime <= '2001-12-08 01:04:00'
AND Value = 1
AND wwEdgeDetection = 'Both'
```

The results are:

DateTime	TagName	Value
2001-12-08 00:01:00.000	SysPulse	1
2001-12-08 00:01:00.000	MyPulse	1
2001-12-08 00:01:40.000	MyPulse	1
2001-12-08 00:02:00.000	SysPulse	1
2001-12-08 00:02:20.000	MyPulse	1
2001-12-08 00:03:00.000	SysPulse	1
2001-12-08 00:03:00.000	MyPulse	1
2001-12-08 00:03:40.000	MyPulse	1
2001-12-08 00:04:00.000	SysPulse	1

```
SELECT DateTime, SysPulse, MyPulse FROM OpenQuery(INSQL, 'SELECT DateTime,
SysPulse, MyPulse
   FROM WideHistory
      WHERE DateTime >  "2001-12-08 00:59:00"
         AND DateTime <= "2001-12-08 01:04:00"
         AND SysPulse = 1
         AND MyPulse = 1
         AND wwEdgeDetection = "Both"
      ')
```

The results are:

DateTime	SysPulse	MyPulse
2001-12-08 00:01:00.000	1	1
2001-12-08 00:01:40.000	1	1
2001-12-08 00:03:40.000	1	1
2001-12-08 00:04:00.000	1	1

Compare these results with the same query using no edge detection, as shown in the *Edge Detection for Discrete Tags* on page 82.

Predictive Filter

AVEVA Historian supports predictive retrieval. Beginning with AVEVA Historian 2014 R2 Patch 01, the historian can return predictive data based on a "simple linear regression" (SLR) algorithm. More capabilities will be added in future releases.

With AVEVA Historian, you can create a query based on data you have stored to predict additional values in a trend. Historian returns predictive data based on a "simple linear regression" (SLR) algorithm.

For example, based on your currently stored values, you could use the predictive retrieval feature to help predict if a certain production target will be met by the end of the shift. Or, if the Historian loses communication with the data source, you could use predictive retrieval to determine whether and when a tank is likely to become empty.

You can predict:

- Values in between other values.
- Values that extend beyond stored values.

For example, suppose you already captured data for a tag with timestamps up to 3 p.m. on a certain day, but not for the rest of the shift, which ran until 5 p.m., because of a power cut. With predictive retrieval, you can view the interpolated results based between 3 p.m. and 5 p.m. These results are based on the data you received through 3 p.m.

The following is an example of a query that retrieves stored values and reports both those values and additional predictive data:

```
SELECT DateTime, Value, wwFilter
FROM History
   WHERE TagName = 'Tag1'
      AND DateTime >= '2014-01-01 0:00:00.000'
      and DateTime < '2014-01-01 1:00:00.000'
      and wwFilter = 'SLR()'
```

In this example, "SLR" stands for "simple linear regression," the algorithm used by AVEVA Historian to analyze currently stored values and predict other values within the detected trend.

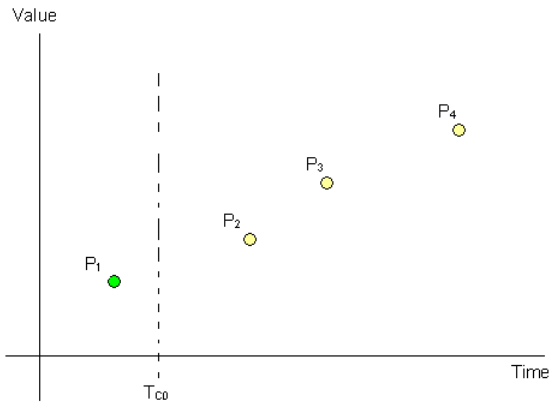
Bounding Value Retrieval

The bounding value retrieval mode returns either the start bound point or the end bound point for a requested point in time. For a start bound point, Historian retrieves the first value on or before the requested date/time. For an end bound point, Historian retrieves the first value after the requested date/time.

If no time is specified, Historian returns the bounding point at the current time.

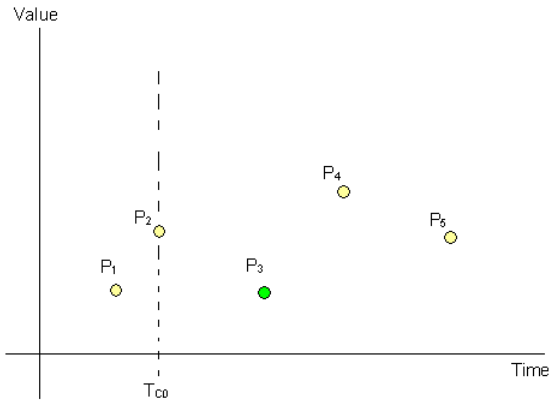
Bounding Value Retrieval - How It Works

The following illustration shows how bounding value retrieval returns a start bound point:



In this case, Historian retrieves the first point on or before the datetime requested in the query. The line (T_{C0}) is the timestamp for which the start bound point is requested. P_1 is returned because that is the start or first point for the query date time.

Historian can also use bounding value retrieval to return an end bound point, as in the following illustration:



In this case, Historian returns first point after the datetime requested in the query. T_{C0} is the timestamp for which the end bound point is requested and P_3 is returned as the ending bound point because this is the first point after the query date time.

Bounding Value Retrieval - Query Examples

You can use the Bounding Value retrieval mode to return a start bound point or an end bound point for a specified date and time. If no time is specified, Historian returns the bounding point at the current time.

To return a start bound point, set the following parameter in your query.

```
wwRetrievalMode = 'StartBound'
```

To return an end bound point, set the following parameter in your query.

```
wwRetrievalMode = 'EndBound'
```

Example 1 - Retrieve start bound point

```
select DateTime,TagName,Value
where TagName 'Plant2.R31.BatchNum'
and wwRetrievalMode = 'StartBound'
and DateTime >= '2019-04-24 12:00:00'
```

The results are:

DateTime	TagName	Value
2019-04-24 11:53:08.5430000	Plant2.R31.BatchNum	912

Example 2 - Retrieve end bound point

```
select DateTime,TagName,Value
where TagName in 'Plant2.R31.BatchNum'
and wwRetrievalMode = 'EndBound'
and DateTime >= '2019-04-24 12:00:00'
```

The results are:

DateTime	TagName	Value
2019-04-24 14:11:13.3840000	Plant2.R31.BatchNum	926

Understanding Retrieval Options

In all retrieval modes, you can adjust which values the historian returns by specifying retrieval options. The retrieval options are implemented as special parameters that you set as part of the retrieval query. This section explains each of these options. For an overview of which options apply to which retrieval modes, see *Which Options Apply to Which Retrieval Modes?* on page 89.

Which Options Apply to Which Retrieval Modes?

The following table shows which retrieval options apply to which modes. If you specify an option in a mode where it isn't applicable, the option is ignored.

	Cycle Count (X Values over Equal Time Intervals) (wwCycleCount) on page 91	*Resolution (Values Spaced Every X ms) (wwResolution)* on page 93	*Time Deadband (wwTimeDeadband)* on page 97	*Value Deadband (wwValueDeadband)* on page 101	*History Version (wwVersion)* on page 104	*Interpolation Type (wwInterpolationType)* on page 105	*TimeStamp Rule (wwTimeStampRule)* on page 107	*Quality Rule (wwQualityRule)* on page 111	*State Calculation (wwStateCalc)* on page 119	*Analog Value Filtering (wwFilter)* on page 120	*Selecting Values for Analog Summary Tags (wwValueSelector)* on page 125	*Predictive Retrieval (wwFilter)* (see "*Predictive Filter*" on page 86)**
Cyclic Retrieval on page 27	Y	Y			Y		Y*	Y		Y	Y	
Delta Retrieval on page 30			Y	Y	Y			Y		Y	Y	Y
Full Retrieval on page 35					Y			Y		Y	Y	Y
Interpolated Retrieval on page 37	Y	Y			Y	Y	Y	Y		Y	Y	
Best Fit Retrieval (see "*Best Fit Retrieval*" on page 41)	Y	Y			Y	Y		Y		Y	Y	Y
Average Retrieval on page 46	Y	Y			Y	Y	Y	Y		Y	Y	
Minimum Retrieval on page 50	Y	Y			Y			Y		Y	Y	
Maximum Retrieval on page 55	Y	Y			Y			Y		Y	Y	
Integral Retrieval on page 59	Y	Y			Y	Y	Y	Y		Y	Y	

	Cycle Count (X Values over Equal Time Intervals) (wwCycleCount) on page 91	*Resolution (Values Spaced Every X ms) (wwResolution)* on page 93	*Time Deadband (wwTimeDeadband)* on page 97	*Value Deadband (wwValueDeadband)* on page 101	*History Version (wwVersion)* on page 104	*Interpolation Type (wwInterpolationType)* on page 105	*TimeStamp Rule (wwTimeStampRule)* on page 107	*Quality Rule (wwQualityRule)* on page 111	*State Calculation (wwStateCalc)* on page 119	*Analog Value Filtering (wwFilter)* on page 120	*Selecting Values for Analog Summary Tags (wwValueSelector)* on page 125	*Predictive Retrieval (wwFilter)* (see "*Predictive Filter*" on page 86)**
Slope Retrieval on page 62					Y			Y		Y	Y	
Counter Retrieval on page 65	Y	Y			Y		Y	Y		Y		
ValueState Retrieval on page 70	Y	Y			Y		Y	Y	Y	Y		
RoundTrip Retrieval on page 75	Y	Y			Y		Y	Y	Y	Y		
Bounding Values Retrieval (see "*Bounding Value Retrieval*" on page 87)												

* - only on AVEVA Historian 9.0 and later
** - only AVEVA Historian 2014 R2 P01 and later

Using Retrieval Options in a Transact-SQL Statement

You can retrieve data in the AVEVA Historian extension tables using normal Transact-SQL code, as well as the specialized SQL time domain extensions provided by the AVEVA Historian. The AVEVA Historian extensions provide an easy way to query time-based data from the history tables. They also provide additional functionality not supported by Transact-SQL.

Note: The wwParameters extension is reserved for future use. The wwRowCount parameter is still supported, but is deprecated in favor of wwCycleCount.

The extensions are implemented as "virtual" columns in the extension tables. When you query an extension table, you can specify values for these column parameters to manipulate the data that will be returned. You will need to specify any real-time extension parameters each time that you execute the query.

For example, you could specify a value for the wwResolution column in the query so that a resolution is applied to the returned data set:

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-02 10:00:00'
         AND DateTime <= '2001-12-02 10:02:00'
         AND Value >= 50
         AND wwResolution = 10
         AND wwRetrievalMode = 'cyclic'
```

Because the extension tables provide additional functionality that is not possible in a normal SQL Server, certain limitations apply to the Transact-SQL supported by these tables. For more information, see *Unsupported or Limited Syntax Options* on page 18.

Although the Microsoft SQL Server may be configured to be case-sensitive, the values for the virtual columns in the extension tables are always case-insensitive.

Note: You cannot use the IN clause or OR clause to specify more than one condition for a time domain extension. For example, "`wwVersion IN ('original', 'latest')`" and "`wwRetrievalMode = 'Delta' OR wwVersion = 'latest'`" are not supported.

For general information on creating SQL queries, see your Microsoft SQL Server documentation.

Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)

In retrieval modes that use cycles, the cycle count determines the number of cycles for which data is retrieved. The number of actual return values is not always identical with this cycle count. In "truly cyclic" modes (Cyclic, Interpolated, Average, and Integral), a single data point is returned for every cycle boundary. However, in other cycle-based modes (Best Fit, Minimum, Maximum, Counter, ValueState, and RoundTrip), multiple or no data points may be returned for a cycle, depending on the nature of the data.

The length of each cycle (the "resolution" of the returned values) is calculated as follows:

$$D_C = D_Q / (n - 1)$$

Where D_C is the length of the cycle, D_Q is the duration of the query, and n is the cycle count.

Instead of specifying a cycle count, you can specify the resolution. In that case, the cycle count is calculated based on the resolution and the query duration. For more information, see *Resolution (Values Spaced Every X ms) (wwResolution)* on page 93.

This option is relevant in the following retrieval modes:

- *Cyclic Retrieval* on page 27
- *Interpolated Retrieval* on page 37
- *"Best Fit" Retrieval* (see "*Best Fit Retrieval*" on page 41)
- *Average Retrieval* on page 46
- *Minimum Retrieval* on page 50
- *Maximum Retrieval* on page 55
- *Integral Retrieval* on page 59
- *Counter Retrieval* on page 65
- *ValueState Retrieval* on page 70
- *RoundTrip Retrieval* on page 75

The application of the cycle count also depends on whether you are querying a wide table. If you are querying the History table, the cycle count determines the number of rows returned per tag. For example, a query that applies a cycle count of 20 to two tags will return 40 rows of data (20 rows for each tag). If you are querying a WideHistory table, the cycle count specifies the total number of rows to be returned, regardless of how many tags were queried. For example, a query that applies a cycle count of 20 to two tags returns 20 rows of data.

Values chosen:

- If wwResolution and wwCycleCount are not specified, then a default of 100 cycles are chosen.

- If wwResolution and wwCycleCount are set to 0, then a default of 100000 cycles are chosen.
- If wwResolution and wwCycleCount are both set, then wwCycleCount is ignored.
- If wwCycleCount is specified and is less than 0, then a default of 100 cycles are chosen.
- For ValueState retrieval, if the start time of the cycle is excluded, no states are returned for the first cycle.
- For ValueState retrieval, if the end time of the cycle is excluded, no states are returned for the last cycle.

Cycle Count - Query Examples

See the following examples of queries that demonstrate the cycle count behavior if applied to a single tag or to multiple tags in the same query:

- *Cycle Count - Query 1: Using a Single Tag* on page 92
- *Cycle Count - Query 2: Using Multiple Tags* on page 92

Cycle Count - Query 1: Using a Single Tag

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime < '2001-12-09 11:36'
         AND wwRetrievalMode = 'Cyclic'
         AND wwCycleCount = 300
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:00.200	SysTimeSec	0
2001-12-09 11:35:00.400	SysTimeSec	0
2001-12-09 11:35:00.600	SysTimeSec	0
...		
2001-12-09 11:35:59.200	SysTimeSec	59
2001-12-09 11:35:59.400	SysTimeSec	59
2001-12-09 11:35:59.600	SysTimeSec	59
2001-12-09 11:35:59.800	SysTimeSec	59

Cycle Count - Query 2: Using Multiple Tags

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysTimeMin','SysTimeSec')
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime < '2001-12-09 11:36'
         AND wwRetrievalMode = 'Cyclic'
         AND wwCycleCount = 300
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeMin	35
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:00.200	SysTimeMin	35
2001-12-09 11:35:00.200	SysTimeSec	0
2001-12-09 11:35:00.400	SysTimeMin	35
2001-12-09 11:35:00.400	SysTimeSec	0
2001-12-09 11:35:00.600	SysTimeMin	35
2001-12-09 11:35:00.600	SysTimeSec	0
...		
2001-12-09 11:35:59.200	SysTimeMin	35
2001-12-09 11:35:59.200	SysTimeSec	59
2001-12-09 11:35:59.400	SysTimeMin	35
2001-12-09 11:35:59.400	SysTimeSec	59
2001-12-09 11:35:59.600	SysTimeMin	35
2001-12-09 11:35:59.600	SysTimeSec	59
2001-12-09 11:35:59.800	SysTimeMin	35
2001-12-09 11:35:59.800	SysTimeSec	59

Notice that the values of the two tags are mixed together in the same column.

Resolution (Values Spaced Every X ms) (wwResolution)

In retrieval modes that use cycles, the resolution is the sampling interval for retrieving data, that is, the length of each cycle.

The number of cycles, therefore, depends on the time period and the resolution:

number of cycles = time period / resolution

The number of actual return values is not always identical with this cycle count. In "truly cyclic" modes (Cyclic, Interpolated, Average, and Integral), a single data point is returned for every cycle boundary. However, in other cycle-based modes (Best Fit, Minimum, Maximum, Counter, and ValueState), multiple or no data points may be returned for a cycle, depending on the nature of the data.

Note: The rowset is guaranteed to contain one row for each tag in the normalized query at every resolution interval, regardless of whether a physical row exists in history at that particular instance in time. The value contained in the row is the last known physical value in history, at that instant, for the relevant tag.

Instead of specifying a resolution, you can specify the cycle count directly. In that case, the resolution is calculated based on the cycle count and the query duration. For more information, see *Cycle Count (X Values over Equal Time Intervals) (wwCycleCount)* on page 91.

This option is relevant in the following retrieval modes:

- *Cyclic Retrieval* on page 27

- *Interpolated Retrieval* on page 37
- *"Best Fit" Retrieval* (see "*Best Fit Retrieval*" on page 41)
- *Average Retrieval* on page 46
- *Minimum Retrieval* on page 50
- *Maximum Retrieval* on page 55
- *Integral Retrieval* on page 59
- *Counter Retrieval* on page 65
- *ValueState Retrieval* on page 70
- *RoundTrip Retrieval* on page 75

For delta retrieval, you can achieve similar results by using a time deadband. For more information, see *Time Deadband (wwTimeDeadband)* on page 97.

Resolution - Query Example

The following query returns rows that are spaced at 2 sec (2000 msec) intervals over the requested time period. Data is retrieved cyclically.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysTimeMin','SysTimeSec')
          AND DateTime >= '2001-12-09 11:35'
          AND DateTime <= '2001-12-09 11:36'
          AND wwRetrievalMode = 'Cyclic'
          AND wwResolution = 2000
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeMin	35
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:02.000	SysTimeMin	35
2001-12-09 11:35:02.000	SysTimeSec	2
2001-12-09 11:35:04.000	SysTimeMin	35
2001-12-09 11:35:04.000	SysTimeSec	4
2001-12-09 11:35:06.000	SysTimeMin	35
...		
2001-12-09 11:35:54.000	SysTimeMin	35
2001-12-09 11:35:54.000	SysTimeSec	54
2001-12-09 11:35:56.000	SysTimeMin	35
2001-12-09 11:35:56.000	SysTimeSec	56
2001-12-09 11:35:58.000	SysTimeMin	35
2001-12-09 11:35:58.000	SysTimeSec	58
2001-12-09 11:36:00.000	SysTimeMin	36

2001-12-09 11:36:00.000	SysTimeSec	0

About Phantom Cycles

The phantom cycle is the name given to the cycle that leads up to the query start time. It is used to calculate which initial value to return at the query start time for all retrieval modes. Some retrieval modes use the phantom cycle to simply find the last known value prior to the query start time, whereas other retrieval modes use the entire cycle to calculate aggregates. The different uses of the phantom cycle can be seen in the following table.

Simple use of phantom cycle	Cycles not defined, but similar simple use of time before query start time	Phantom cycle used to calculate aggregates
Cyclic	Delta	Min
Interpolated	Full	Max
Best Fit	Slope	Average
		Integral
		Counter
		ValueState
		RoundTrip

It's common to expect a single aggregate row returned for a particular time interval. You can accomplish this in several ways.

The following example is querying for hourly averages. It returns a single row time stamped at the query start time. If the query included the query end point by including an equal sign for it, the query would also have returned an additional row at the query end time.

```
SELECT DateTime, Value, Quality, QualityDetail, OPCQuality
   FROM History
      WHERE TagName IN ('SysTimeSec')
         AND DateTime >= '2017-12-12 08:00:00'
         AND DateTime < '2017-12-12 09:00:00'
         AND wwRetrievalMode = 'Avg'
         AND wwResolution = 3600000
```

The results are:

DateTime	Value	Quality	QualityDetail	OPCQuality
2017-12-1208:00:00.0000000	29.5	0	192	192

What may be confusing in this example is the calculation of the average in the returned row for the phantom cycle leading up to the query start time. The query specifies a positive one hour time interval between the query start time and the query end time. You may therefore expect that the calculated and returned average should be for the specified interval.

However, the time difference between start and end time in the above query is not actually required because the resolution is provided explicitly (wwResolution = 36000000). If the query specified an end time equal to the specified start time and if it included the equal sign for the end time, the query would still return the same single row of data.

```
SELECT DateTime, Value, Quality, QualityDetail, OPCQuality
   FROM History
      WHERE TagName IN ('SysTimeSec')
         AND DateTime = '2017-10-16 08:00:00'
         AND DateTime <= '2017-10-16 09:00:00'
         AND wwRetrievalMode = 'Avg'
         AND wwCycleCount = 1
```

The results are:

DateTime	Value	Quality	QualityDetail	OPCQuality
2017-10-16 08:00:00.0000000	29.5	0	192	192

This second example also asks for hourly averages and it also returns only a single row of data stamped at the query start time. This query, however, must specify a time difference between the start and end time, because the resolution is not explicitly defined in the query.

As in the preceding query, the specified interval and cycle count of 1 may look like the returned row has been calculated for the specified interval, but the returned row is once again for the phantom cycle leading up to the start time.

For some queries, you may want to be certain to include values on a cycle boundary. For example, the following query is looking for a minimum value within a cycle. In this query, the beginning DateTime statement uses ">=" to ensure that the entire cycle is queried. Even if the minimum value happens to be at the beginning of the cycle, the following query will provide an accurate result:

```
SELECT StartDateTime, *
   FROM History
      WHERE TagName = 'SysTimeSec'
      AND DateTime >= '2016-03-31 15:41:10'
      AND DateTime < '2016-03-31 15:41:20'
      AND wwRetrievalMode = 'Min'
      AND Quality <> 133
      AND wwCycleCount = 1
```

The StartDateTime makes it easier to see which time interval was used to calculate the returned aggregate. This column returns the time stamp of the beginning of the cycle used for the aggregate calculation. The time stamp is always returned in accordance with the specified time zone and always has the same offset as the time stamp returned in the DateTime column, even when the two time stamps are on different sides of a DST change.

Assuming results are timestamped at the end of the cycle (as is done by default when wwTimeStampRule is set to END), the initial rows in the examples above would return a DateTime equal to '2009-10-16 08:00:00', and the StartDateTime column would return '2009-10-16 07:00:00' making it easy to interpret the result.

If instead the query were to ask for results time stamped at the beginning of the cycle with wwTimeStampRule set to START, the initial rows in the same examples would still return a DateTime equal to '2009-10-16 08:00:00', but the time stamp has now been shifted in accordance with the time stamp request. The result is therefore calculated for the specified time interval between 8 a.m. and 9 a.m. In this example, the new StartDateTime column would return the same time stamp as the DateTime column, '2009-10-16 08:00:00', again making it easier to interpret the result.

For retrieval modes for which cycles are defined, the StartDateTime column returns the cycle start time. These modes are:

- *Cyclic Retrieval* on page 27
- *Interpolated Retrieval* on page 37
- *"Best Fit" Retrieval* (see "*Best Fit Retrieval*" on page 41)
- *Average Retrieval* on page 46
- *Minimum Retrieval* on page 50
- *Maximum Retrieval* on page 55
- *Integral Retrieval* on page 59
- *Counter Retrieval* on page 65
- *ValueState Retrieval* on page 70
- *RoundTrip Retrieval* on page 75

In the remaining retrieval modes, the StartDateTime column returns the same time stamp as the DateTime column.

For an additional example, see *Querying Aggregate Data in Different Ways* on page 171.

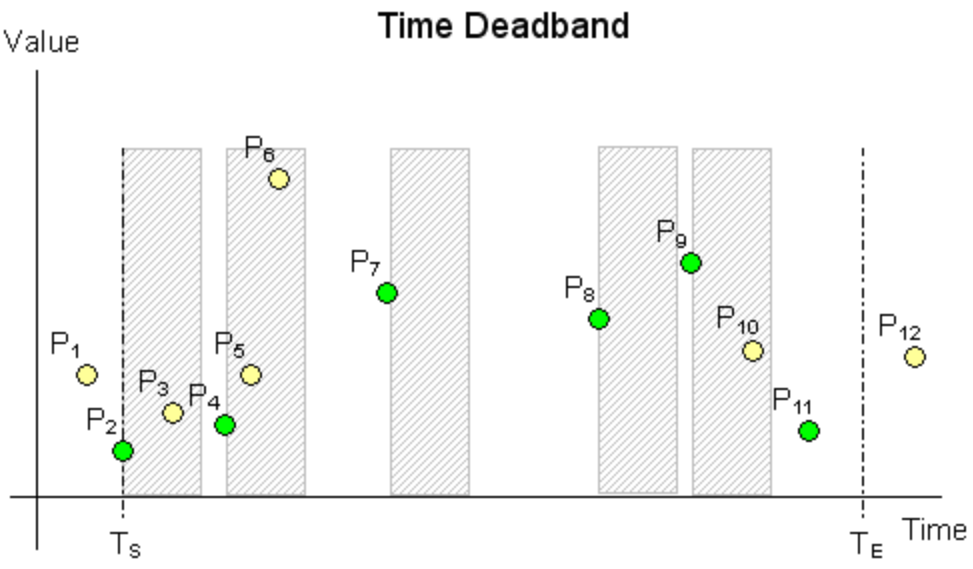
Time Deadband (wwTimeDeadband)

A time deadband for retrieval controls the time resolution of data returned in delta mode. Any value changes that occur within the time deadband are not returned.

Time deadbands can be applied to analog, discrete, and string tags.

The deadband "base value" is reset each time that a value is returned, so that the last value returned acts as the basis for the deadband.

The following illustration shows an example of applying a time deadband:



Data is retrieved for the time period starting with T_S and ending with T_E. All points in the graphic represent data values stored on the historian. The gray areas represent the time deadband, which starts anew with every returned value. Only the green points (P_2, P_4, P_7, P_8, P_9, P_{11}) are returned. The other points are not returned because they fall within a deadband.

Time Deadband - Query Examples

To apply a time deadband, set the wwTimeDeadband parameter in your query.

For examples, see the following:

- *Time Deadband - Query 1* on page 98
- *Time Deadband - Query 2* on page 99
- *Time Deadband - Query 3* on page 100

Note: All of these example queries return data values for the analog tag 'SysTimeSec'.

Time Deadband - Query 1

This query specifies to only return data that changed during a 5 second time deadband.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:37'
         AND wwRetrievalMode = 'Delta'
         AND wwTimeDeadband = 5000
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:06.000	SysTimeSec	6
2001-12-09 11:35:12.000	SysTimeSec	12
2001-12-09 11:35:18.000	SysTimeSec	18
2001-12-09 11:35:24.000	SysTimeSec	24
2001-12-09 11:35:30.000	SysTimeSec	30
2001-12-09 11:35:36.000	SysTimeSec	36
2001-12-09 11:35:42.000	SysTimeSec	42
2001-12-09 11:35:48.000	SysTimeSec	48
2001-12-09 11:35:54.000	SysTimeSec	54
2001-12-09 11:36:00.000	SysTimeSec	0
2001-12-09 11:36:06.000	SysTimeSec	6
2001-12-09 11:36:12.000	SysTimeSec	12
2001-12-09 11:36:18.000	SysTimeSec	18

2001-12-09 11:36:24.000	SysTimeSec	24
2001-12-09 11:36:30.000	SysTimeSec	30
2001-12-09 11:36:36.000	SysTimeSec	36
2001-12-09 11:36:42.000	SysTimeSec	42
2001-12-09 11:36:48.000	SysTimeSec	48
2001-12-09 11:36:54.000	SysTimeSec	54
2001-12-09 11:37:00.000	SysTimeSec	0

Time Deadband - Query 2

This query specifies to only return data that changed during a 4900 millisecond time deadband.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:37'
         AND wwRetrievalMode = 'Delta'
         AND wwTimeDeadband = 4900
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:05.000	SysTimeSec	5
2001-12-09 11:35:10.000	SysTimeSec	10
2001-12-09 11:35:15.000	SysTimeSec	15
2001-12-09 11:35:20.000	SysTimeSec	20
2001-12-09 11:35:25.000	SysTimeSec	25
2001-12-09 11:35:30.000	SysTimeSec	30
2001-12-09 11:35:35.000	SysTimeSec	35
2001-12-09 11:35:40.000	SysTimeSec	40
2001-12-09 11:35:45.000	SysTimeSec	45
2001-12-09 11:35:50.000	SysTimeSec	50
2001-12-09 11:35:55.000	SysTimeSec	55
2001-12-09 11:36:00.000	SysTimeSec	0
2001-12-09 11:36:05.000	SysTimeSec	5
2001-12-09 11:36:10.000	SysTimeSec	10
2001-12-09 11:36:15.000	SysTimeSec	15
2001-12-09 11:36:20.000	SysTimeSec	20
2001-12-09 11:36:25.000	SysTimeSec	25
2001-12-09 11:36:30.000	SysTimeSec	30

2001-12-09 11:36:35.000	SysTimeSec	35
2001-12-09 11:36:40.000	SysTimeSec	40
2001-12-09 11:36:45.000	SysTimeSec	45
2001-12-09 11:36:50.000	SysTimeSec	50
2001-12-09 11:36:55.000	SysTimeSec	55
2001-12-09 11:37:00.000	SysTimeSec	0

Time Deadband - Query 3

This query specifies to only return data that changed during a 2000 millisecond time deadband.

```
SELECT DateTime, TagName, Value
   FROM History
      WHERE TagName IN ('SysTimeSec','SysTimeMin')
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:36'
         AND wwRetrievalMode = 'Delta'
         AND wwTimeDeadband = 2000
```

The results are:

DateTime	TagName	Value
2001-12-09 11:35:00.000	SysTimeSec	0
2001-12-09 11:35:00.000	SysTimeMin	35
2001-12-09 11:35:03.000	SysTimeSec	3
2001-12-09 11:35:06.000	SysTimeSec	6
2001-12-09 11:35:09.000	SysTimeSec	9
2001-12-09 11:35:12.000	SysTimeSec	12
2001-12-09 11:35:15.000	SysTimeSec	15
2001-12-09 11:35:18.000	SysTimeSec	18
2001-12-09 11:35:21.000	SysTimeSec	21
2001-12-09 11:35:24.000	SysTimeSec	24
2001-12-09 11:35:27.000	SysTimeSec	27
2001-12-09 11:35:30.000	SysTimeSec	30
2001-12-09 11:35:33.000	SysTimeSec	33
2001-12-09 11:35:36.000	SysTimeSec	36
2001-12-09 11:35:39.000	SysTimeSec	39
2001-12-09 11:35:42.000	SysTimeSec	42
2001-12-09 11:35:45.000	SysTimeSec	45
2001-12-09 11:35:48.000	SysTimeSec	48
2001-12-09 11:35:51.000	SysTimeSec	51

2001-12-09 11:35:54.000	SysTimeSec	54
2001-12-09 11:35:57.000	SysTimeSec	57
2001-12-09 11:36:00.000	SysTimeSec	0
2001-12-09 11:36:00.000	SysTimeMin	36

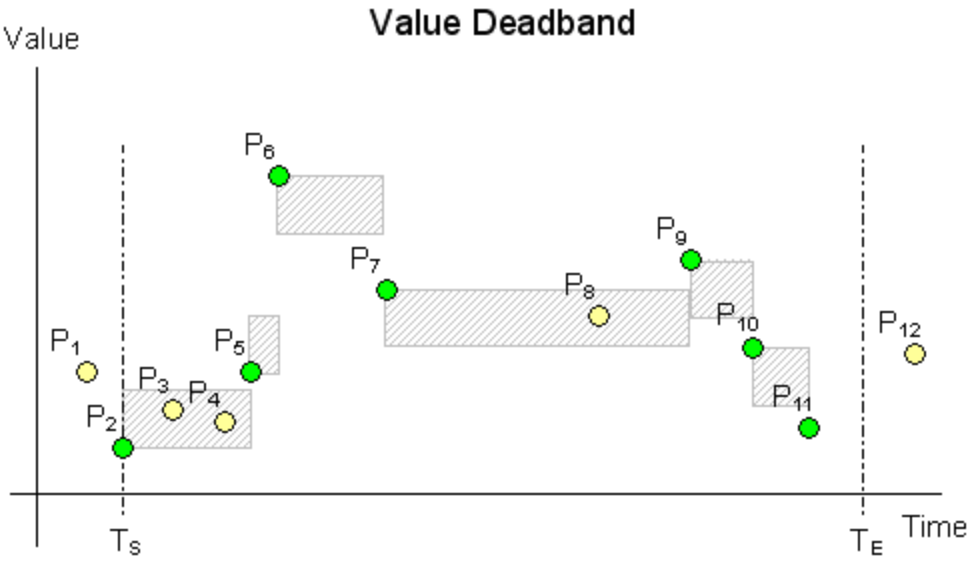
Value Deadband (wwValueDeadband)

A value deadband for retrieval controls the value resolution of data returned in delta mode. Any data values that change less than the specified deadband are not returned. The deadband is a percentage of a tag's full scale in engineering units.

The deadband "base value" is reset each time that a value is returned, so that the last value returned acts as the basis for the deadband.

Changes in quality will force a value to be returned even if the value deadband has not been met.

The following illustration shows an example of applying a value deadband:



Data is retrieved for the time period starting with T_S and ending with T_E. All points in the graphic represent data values stored on the historian. The gray areas represent the value deadband, which starts anew with every returned value. Only the green points (P_2, P_5, P_6, P_7, P_9, P_{10}, P_{11}) are returned. The other points are not returned because they fall within a deadband.

Value Deadband - Query Examples

See the following examples of queries that use value deadband:

- *Value Deadband - Query 1* on page 102
- *Value Deadband - Query 2* on page 102

Note: Each of these examples returns data values for the analog tag 'SysTimeSec'. The minimum engineering unit for 'SysTimeSec' is 0, and the maximum engineering unit is 59.

Value Deadband - Query 1

This query specifies to return only data that changed by more than 10 percent of the tag's full engineering unit range. Using a value deadband of 10 percent equates to an absolute change of 5.9 for 'SysTimeSec.'

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:37'
         AND wwRetrievalMode = 'Delta'
         AND wwValueDeadband = 10
```

The results are:

DateTime	Value
2001-12-09 11:35:00.000	0
2001-12-09 11:35:06.000	6
2001-12-09 11:35:12.000	12
2001-12-09 11:35:18.000	18
2001-12-09 11:35:24.000	24
2001-12-09 11:35:30.000	30
2001-12-09 11:35:36.000	36
2001-12-09 11:35:42.000	42
2001-12-09 11:35:48.000	48
2001-12-09 11:35:54.000	54
2001-12-09 11:36:00.000	0
2001-12-09 11:36:06.000	6
2001-12-09 11:36:12.000	12
2001-12-09 11:36:18.000	18
2001-12-09 11:36:24.000	24
2001-12-09 11:36:30.000	30
2001-12-09 11:36:36.000	36
2001-12-09 11:36:42.000	42
2001-12-09 11:36:48.000	48
2001-12-09 11:36:54.000	54
2001-12-09 11:37:00.000	0

Value Deadband - Query 2

This query specifies to only return data that changed by more than 5 percent of the tag's full engineering unit range. Using a value deadband of 5 percent equates to an absolute change of 2.95 for 'SysTimeSec.'

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-09 11:35'
         AND DateTime <= '2001-12-09 11:37'
         AND wwRetrievalMode = 'Delta'
         AND wwValueDeadband = 5
```

The results are:

DateTime	Value
2001-12-09 11:35:00.000	0
2001-12-09 11:35:03.000	3
2001-12-09 11:35:06.000	6
2001-12-09 11:35:09.000	9
2001-12-09 11:35:12.000	12
2001-12-09 11:35:15.000	15
2001-12-09 11:35:18.000	18
2001-12-09 11:35:21.000	21
2001-12-09 11:35:24.000	24
2001-12-09 11:35:27.000	27
2001-12-09 11:35:30.000	30
2001-12-09 11:35:33.000	33
2001-12-09 11:35:36.000	36
2001-12-09 11:35:39.000	39
2001-12-09 11:35:42.000	42
2001-12-09 11:35:45.000	45
2001-12-09 11:35:48.000	48
2001-12-09 11:35:51.000	51
2001-12-09 11:35:54.000	54
2001-12-09 11:35:57.000	57
2001-12-09 11:36:00.000	0
2001-12-09 11:36:03.000	3
2001-12-09 11:36:06.000	6
2001-12-09 11:36:09.000	9
2001-12-09 11:36:12.000	12
2001-12-09 11:36:15.000	15
2001-12-09 11:36:18.000	18
2001-12-09 11:36:21.000	21
2001-12-09 11:36:24.000	24

2001-12-09 11:36:27.000	27
2001-12-09 11:36:30.000	30
2001-12-09 11:36:33.000	33
2001-12-09 11:36:36.000	36
2001-12-09 11:36:39.000	39
2001-12-09 11:36:42.000	42
2001-12-09 11:36:45.000	45
2001-12-09 11:36:48.000	48
2001-12-09 11:36:51.000	51
2001-12-09 11:36:54.000	54
2001-12-09 11:36:57.000	57
2001-12-09 11:37:00.000	0

History Version (wwVersion)

The AVEVA Historian allows you to overwrite a stored tag value with later versions of the value. The original version of the value is still maintained, so that effectively, multiple versions of the tag value exist at the same point in time.

When retrieving data, you can specify whether to retrieve the originally stored version or the latest version that is available. To do this, set the history version option to "Original" for the original version or "Latest" for the latest available version. If you do not specify the version, the latest version is returned.

To distinguish between a latest value and an original value, the historian returns a special QualityDetail value of 202 for a latest point with good quality.

This option is relevant in all retrieval modes.

History Version - Query Example

This example illustrates using history version. First, consider this query:

```
SELECT TagName, DateTime, Value, wwVersion
   FROM History
      WHERE TagName IN ('SysTimeHour', 'SysTimeMin')
         AND DateTime >= '2001-12-20 0:00'
         AND DateTime <= '2001-12-20 0:05'
         AND wwRetrievalMode = 'Delta'
         AND wwVersion = 'Original'
```

The results are:

TagName	DateTime	Value	wwVersion
SysTimeMin	2001-12-20 00:00:00.000	0	ORIGINAL
SysTimeHour	2001-12-20 00:00:00.000	0	ORIGINAL
SysTimeMin	2001-12-20 00:01:00.000	1	ORIGINAL
SysTimeMin	2001-12-20 00:02:00.000	2	ORIGINAL

SysTimeMin	2001-12-20 00:03:00.000	3	ORIGINAL
SysTimeMin	2001-12-20 00:04:00.000	4	ORIGINAL
SysTimeMin	2001-12-20 00:05:00.000	5	ORIGINAL

When retrieving the latest version, the wwVersion parameter always returns with a value of LATEST for all values, even though many of the values may actually be the original values that came from the I/O Server. To distinguish between an actual latest value and an original value, a special QualityDetail of 202 is returned for a good, latest point.

For example:

```
SELECT DateTime, Value, Quality, QualityDetail, OPCQuality, wwVersion
    FROM History
      WHERE TagName IN ('PV')
        AND DateTime >= '2005-04-17 11:35:00'
        AND DateTime <= '2005-04-17 11:36:00'
        AND wwRetrievalMode = 'Delta'
        AND wwVersion = 'Latest'
```

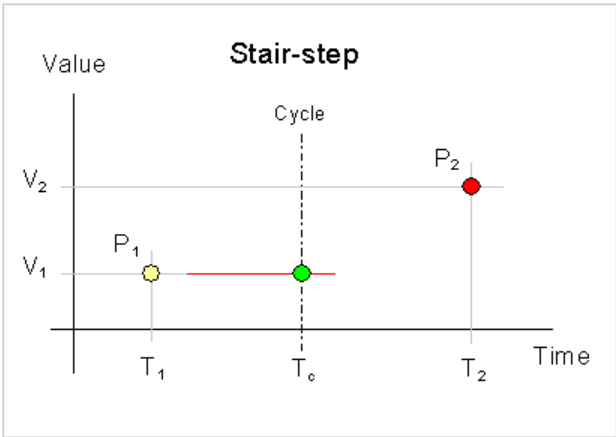
The results are:

DateTime	Value	Quality	QualityDetail	OPCQuality	wwVersion
2005-04-17 11:35:00.000	12.5	0	192	192	LATEST
2005-04-17 11:35:15.000	17.3	0	192	192	LATEST
2005-04-17 11:35:30.000	34.0	0	202	192	LATEST
2005-04-17 11:35:45.000	43.1	0	192	192	LATEST
2005-04-17 11:36:00.000	51.2	0	192	192	LATEST

Interpolation Type (wwInterpolationType)

For various retrieval modes, you can control how analog tag values at cycle boundaries are calculated if there is no actual value stored at that point in time. The options are as follows:

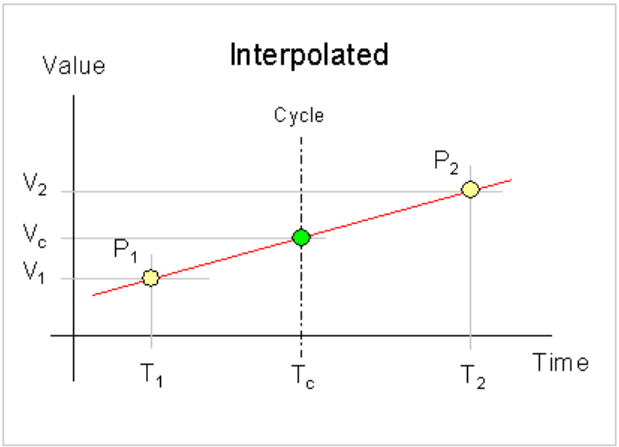
- **Stairstep:** No interpolation occurs. The value at the cycle boundary is assumed to be the same value as the last stored value before the cycle boundary. The last known point is returned with the given cycle time. If no valid value can be found, a NULL is returned.



- **Linear:** The historian calculates a new value at the cycle boundary by interpolating between the last stored value before the boundary and the first stored value after the boundary. If either of these values is NULL, it returns the last stored value before the boundary.

  Expressed as a formula, V_c is calculated as:

  $V_c = V_1 + ((V_2 - V_1) * ((T_c - T_1) / (T_2 - T_1)))$



The type of data that you want to retrieve usually determines the interpolation type to use. For example, if you have a thermocouple, the temperature change is linear, so it's best to use linear interpolation. If you have a tag that contains discrete measurements, such as a set point, then you probably want to use stair-stepped values. In general, it is recommended that you use linear interpolation as the general setting, and use stair-stepped values for the exceptions.

This option is relevant in the following retrieval modes:

- -old-Interpolated Retrieval
- "Best Fit" Retrieval
- *Average Retrieval* on page 46
- *Integral Retrieval* on page 59

The quality of an interpolated point is determined by the wwQualityRule setting. For more information, see *Quality Rule (wwQualityRule)* on page 111.

The interpolation type can be set on three levels:

- The AVEVA Historian system-wide setting. The system-wide setting must be either stair-step or interpolated. For more information, see "System Parameters" on page 36. This setting is configured using the AVEVA Historian Configuration Editor.
- The individual analog tag setting. You can configure an individual analog tag to use the system-wide setting or either stair-stepped values or linear interpolation. The individual tag setting will override the system-wide setting. This setting is configured using the AVEVA Historian Configuration Editor.
- The setting for the wwInterpolationType parameter in the query. This setting overrides any other setting for all tags in the query.

The wwInterpolationType parameter is dynamically used both for input for the query, when you need to override the individual tag settings, and for output for each individual row to show whether a particular row value was calculated using linear interpolation (returned as "LINEAR") or if it is a stair-stepped value (returned as "STAIRSTEP").

To force a query to always use linear interpolation whenever applicable, specify the following in the query:

```
AND wwInterpolationType = 'Linear'
```

To force a query to always return stair-stepped values, specify the following in the query:

```
AND wwInterpolationType = 'StairStep'
```

TimeStamp Rule (wwTimeStampRule)

For various cycle-based retrieval modes, you can control whether the returned values are timestamped at the beginning or at the end of each cycle.

To force a query to timestamp results at the start of a cycle, specify the following in the query:

```
AND wwTimeStampRule = 'Start'
```

To force a query to timestamp results at the end of a cycle, specify the following in the query:

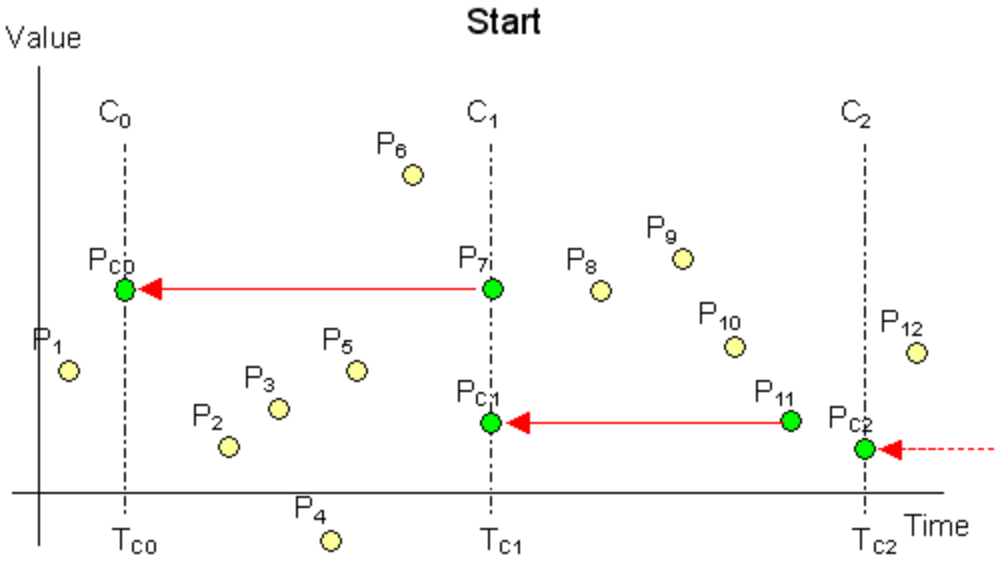
```
AND wwTimeStampRule = 'End'
```

If you include the wwTimeStampRule column in your SELECT statement, it will show which timestamp rule has been applied for the individual row, if applicable.

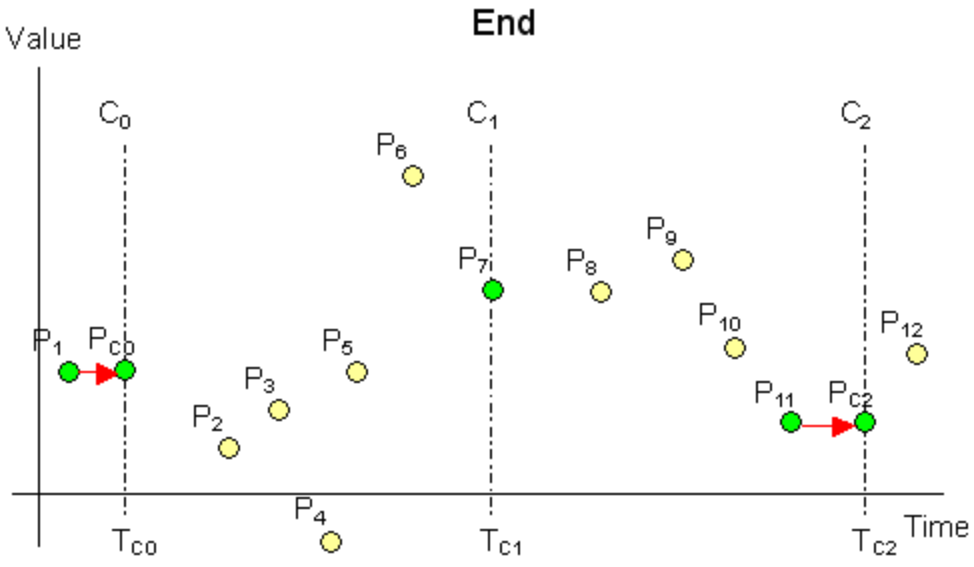
The options are as follows:

- **Start:** The value for a given cycle is stamped with the cycle start time. For example, in the following illustration of a cyclic query, the following values are returned at the cycle boundaries:
  - At T_{C0}: P_7, because it falls on the cycle boundary. In cyclic mode, if there is a value right on the cycle boundary, it is considered to belong to the cycle before the boundary. In this case, this is the cycle starting at T_{C0} and ending at T_{C1}, and because the Start timestamp rule is used, the value is timestamped at T_{C0}.
  - At T_{C1}: P_{11}, because it is the last value in the cycle starting at T_{C1} and ending at T_{C2}

- At T_{C2}: The last value in the "phantom" cycle starting at T_{C2}



- **End:** The value for a given cycle is stamped with the cycle end time. For example, in the following illustration of a cyclic query, the following values are returned at the cycle boundaries:
  - At T_{C0}: P_1, because it is the last value in the "phantom" cycle ending at T_{C0}. Because the End timestamp rule is used, the value is timestamped at T_{C0}.
  - At T_{C1}: P_7, because it falls on the cycle boundary. In cyclic mode, if there is a value right on the cycle boundary, it is considered to belong to the cycle before the boundary. In this case, this is the cycle starting at T_{C0} and ending at T_{C1}, and because the End timestamp rule is used, the value is timestamped at T_{C1}.
  - At T_{C2}: P_{11}, because it is the last value in the cycle ending at T_{C2}


End
Value
C0
C1
C2
P6
P7
P8
P9
P10
P12
P1
Pc0
P5
P3
P2
P11
Pc2
P4
Tc0
Tc1
Tc2
Time

- **Server default:** Either Start or End is used, depending on the system parameter setting on the AVEVA Historian.

This option is relevant in the following retrieval modes:

- *Cyclic Retrieval* on page 27 (AVEVA Historian 9.0 and later)
- *Interpolated Retrieval* on page 37
- *Average Retrieval* on page 46
- *Integral Retrieval* on page 59

- *Counter Retrieval* on page 65
- *ValueState Retrieval* on page 70
- *RoundTrip Retrieval* on page 75

Time Zone (wwTimeZone)

For AVEVA Historianversion 8.0 and later, all history data is stored in Coordinated Universal Time (UTC). The wwTimeZone extension allows you to specify the time zone to be used for the timestamps of the returned data values. The Retrieval subsystem will convert the timestamps to local time in the specified time zone.

The wwTimeZone extension may be assigned any of the values stored in the TimeZone column of the TimeZone table in the Runtime database.

The TimeZone table is repopulated at every system startup from Microsoft operating system registry entries, and will therefore reflect the time zones available from the server operating system, including any new or custom time zones which might be added by operating system service packs or installed software.

The Retrieval subsystem will automatically correct for daylight savings time in the requested time zone. When computing daylight savings and time zone parameters, the settings of the server operating system are used. The Retrieval subsystem does not provide any means for using client-side settings.

If wwTimeZone is not specified, the time zone for retrieval defaults to the time zone of the AVEVA Historian computer.

For example:

```
SELECT TagName, DateTime, Value, wwTimeZone
   FROM History
      WHERE TagName IN ('SysTimeHour', 'SysTimeMin')
         AND DateTime >= '2001-12-20 0:00'
         AND DateTime <= '2001-12-20 0:05'
         AND wwRetrievalMode = 'Delta'
         AND wwTimeZone = 'W. Europe Standard Time'
```

The results are:

TagName	DateTime	Value	wwTimeZone
SysTimeMin	2001-12-20 00:00:00.000	0	W. Europe Standard Time
SysTimeHour	2001-12-20 00:00:00.000	15	W. Europe Standard Time
SysTimeMin	2001-12-20 00:01:00.000	1	W. Europe Standard Time
SysTimeMin	2001-12-20 00:02:00.000	2	W. Europe Standard Time
SysTimeMin	2001-12-20 00:03:00.000	3	W. Europe Standard Time
SysTimeMin	2001-12-20 00:04:00.000	4	W. Europe Standard Time
SysTimeMin	2001-12-20 00:05:00.000	5	W. Europe Standard Time

Quality Rule (wwQualityRule)

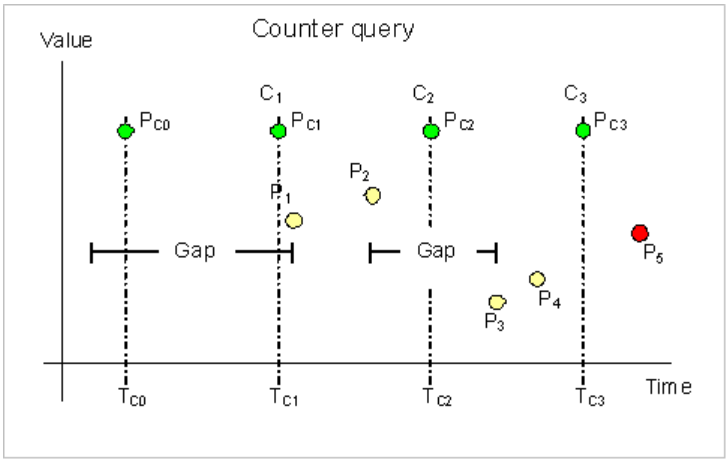
The quality rule can be used to specify whether values with certain characteristics are explicitly excluded from consideration by data retrieval. This parameter will override the setting of the QualityRule system parameter. Valid values are GOOD, EXTENDED, or OPTIMISTIC.

- A quality rule of GOOD means that data values with uncertain (64) OPC quality are not used in the retrieval calculations and are ignored. Values with bad QualityDetail indicate gaps in the data.
- A quality rule of EXTENDED means that data values with both good and uncertain OPC quality are used in the retrieval calculations. Values with bad QualityDetail indicate gaps in the data.
- A quality rule of OPTIMISTIC means that calculations that include some good and some NULL values do not cause the overall calculations to return NULL.

You can apply a quality rule to all retrieval modes.

The OPTIMISTIC setting for the quality rule lets you retrieve information that is possibly incomplete but may nevertheless provide better results where the calculation cycle contains data gaps. This setting calculates using the last known good value prior to the gap (if possible). The logic for determining the quality of the points returned remains unchanged. The integral retrieval mode is an exception to this where the integral is scaled up to cover gaps. For more information, see *Integral Retrieval* on page 59.

The following figure shows a counter retrieval situation in which three of the four shown cycle boundaries are located in data gaps. Without using OPTIMISTIC, counter queries would return a NULL at all cycle boundaries because the mode needs valid good values at each end of the cycle calculate a precise difference.



If the query were to specify OPTIMISTIC, the counter mode will always return rows with numeric counter values and good quality. These rows may or may not be precise. The PercentGood column of the row returns the percentage of time in each cycle in which retrieval was able to find values stored with good quality, so if the PercentGood is anything less than 100, then the returned row may be incorrect. Quality is returned as uncertain if percent good is not 100 percent.

Now look at the counter values that are returned using OPTIMISTIC quality in the preceding illustration. The query skips the value to be returned at the first cycle boundary, because there is not enough information about the cycle prior to that boundary. At the second cycle boundary, the value 0 will be returned, because there was a gap in the data for the entire first cycle. In the second cycle, there are two points, P1 and P2. The query uses P2 as the end value of the cycle and infers a start value of the cycle from P1. At the third cycle boundary, T_{c2}, the query returns P2 – P1. Similarly, at the last cycle boundary, the query returns P4 – P3.

For the integral retrieval mode, the query does not summarize data for gaps because there is no way to know which value to use for the summarization. However, if the query specifies OPTIMISTIC quality, the query uses the last known good value for the summarization in the gap. As described for the counter retrieval example, the PercentGood column also expresses the quality of the calculated value in integral retrieval, so if the PercentGood is anything less than 100, then the returned row may be incorrect.

Quality Rule - Query Examples

To force a query to exclude points with doubtful OPC quality, specify the following in the query:

```
AND wwQualityRule = 'Good'
```

To force a query to use points with both good and doubtful OPC quality, specify the following in the query:

```
AND wwQualityRule = 'Extended'
```

For examples, see the following:

- *Quality Rule - Query 1* on page 113
- *Quality Rule - Query 2* on page 114
- *Quality Rule - Query 3* on page 114
- *Quality Rule - Query 4* on page 115
- *Quality Rule - Query 5* on page 116
- *Quality Rule - Query 6* on page 117
- *Quality Rule - Query 7* on page 117
- *Quality Rule - Query 8* on page 118

Quality Rule - Query 1

If you include the wwQualityRule column in a SELECT statement, it will show which quality rule was used for the individual row, if applicable.

You can combine OPC qualities in a query. For example, if you combine a mixture of good OPC qualities (such as 192 to 219), a good OPC quality (192) will be returned as a combined result.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, wwRetrievalMode
   FROM History
          WHERE TagName = 'I0R5'
          AND DateTime >= '2009-09-12 00:20'
          AND DateTime <= '2009-09-12 00:40'
          AND wwResolution = 10000
          AND wwRetrievalMode = 'Avg'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
I0R5	2009-09-12 00:07	2	193
I0R5	2009-09-12 00:14	3	195
I0R5	2009-09-12 00:22	0	196
I0R5	2009-09-12 00:25	1	199
I0R5	2009-09-12 00:27	0	200
I0R5	2009-09-12 00:29	2	207
I0R5	2009-09-12 00:33	3	215
I0R5	2009-09-12 00:36	0	216
I0R5	2009-09-12 00:39	1	219

The results are:

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode
I0R5	2009-09-12 00:20	2.6	192	192	AVERAGE
I0R5	2009-09-12 00:30	1.0	192	192	AVERAGE
I0R5	2009-09-12 00:40	1.6	192	192	AVERAGE

Quality Rule - Query 2

Similar to *Quality Rule - Query 1* on page 113, if you combine a mixture of doubtful OPC qualities, a doubtful OPC quality (64) will be returned as the combined OPC quality.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, wwRetrievalMode
   FROM History
          WHERE TagName = 'I0R5'
          AND DateTime >= '2009-09-12 00:20'
          AND DateTime <= '2009-09-12 00:40'
          AND wwResolution = 10000
          AND wwRetrievalMode = 'Integral'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
I0R5	2009-09-12 00:07	2	65
I0R5	2009-09-12 00:14	3	68
I0R5	2009-09-12 00:22	0	71
I0R5	2009-09-12 00:25	1	74
I0R5	2009-09-12 00:27	0	79
I0R5	2009-09-12 00:29	2	80
I0R5	2009-09-12 00:33	3	88
I0R5	2009-09-12 00:36	0	92
I0R5	2009-09-12 00:39	1	64

The results are:

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode
I0R5	00:20	26.0	64	64	INTEGRAL
I0R5	00:30	10.0	64	64	INTEGRAL
I0R5	00:40	16.0	64	64	INTEGRAL

Quality Rule - Query 3

When you combine the same OPC quality then that OPC quality will be returned. However, when there is no good point in a cycle for cyclic modes such as Integral, Average, Counter, or AnalogSummary, the returned NULL value will have an OPC quality of 0 and a Quality Detail of 65536, regardless of combined qualities.

```
SELECT TagName, StartDateTime, EndDateTime, OPCQuality, PercentGood,
wwRetrievalMode, first
```

```
FROM AnalogSummaryHistory
      WHERE TagName = 'F0R5'
      AND StartDateTime >= '2009-09-12 00:20'
      AND EndDateTime <= '2009-09-12 00:40'
      AND wwResolution = 10000
      AND wwRetrievalMode = 'Cyclic'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
F0R5	2009-09-12 00:07	1.6	78
F0R5	2009-09-12 00:14	3.1	78
F0R5	2009-09-12 00:22	0.2	78
F0R5	2009-09-12 00:25	0.8	78
F0R5	2009-09-12 00:27	0.4	78
F0R5	2009-09-12 00:29	2.2	78
F0R5	2009-09-12 00:33	3.3	78
F0R5	2009-09-12 00:36	0.3	78
F0R5	2009-09-12 00:39	1.2	78

The results are:

Tagname	StartDate Time	EndDate Time	OPCQuality	Percent Good	wwRetrievalMode	first
F0R5	2009-09-12 00:20	2009-09-12 00:30	78	100	CYCLIC	0.200
F0R5	2009-09-12 00:30	2009-09-12 00:40	78	100	CYCLIC	3.300

Quality Rule - Query 4

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, wwRetrievalMode
   FROM History
      WHERE TagName = 'F0R5'
      AND DateTime >= '2009-09-12 00:20'
      AND DateTime <= '2009-09-12 00:40'
      AND wwResolution = 10000
      AND wwRetrievalMode = 'Avg'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
F0R5	2009-09-12 00:07	1.6	15
F0R5	2009-09-12 00:14	3.1	15
F0R5	2009-09-12 00:22	0.2	15
F0R5	2009-09-12 00:25	0.8	15
F0R5	2009-09-12 00:27	0.4	15

Tagname	DateTime	Resolution	QualityDetail
F0R5	2009-09-12 00:29	2.2	15
F0R5	2009-09-12 00:33	3.3	15
F0R5	2009-09-12 00:36	0.3	15
F0R5	2009-09-12 00:39	1.2	15

The results are:

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode
F0R5	2009-09-12 00:20	NULL	65536	0	AVERAGE
F0R5	2009-09-12 00:30	NULL	65536	0	AVERAGE
F0R5	2009-09-12 00:40	NULL	65536	0	AVERAGE

Quality Rule - Query 5

When you combine a mixture of good, bad, and uncertain OPC qualities, a doubtful OPC quality (64) will be returned as a combined result.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, wwRetrievalMode
   FROM History
      WHERE TagName = 'F0R5'
      AND DateTime >= '2009-09-12 00:20'
      AND DateTime <= '2009-09-12 00:40'
      AND wwResolution = 10000
      AND wwRetrievalMode = 'Avg'
      AND wwQualityRule = 'Optimistic'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
F0R5	2009-09-12 00:07	1.6	15
F0R5	2009-09-12 00:14	3.1	69
F0R5	2009-09-12 00:22	0.2	78
F0R5	2009-09-12 00:25	0.8	200
F0R5	2009-09-12 00:27	0.4	15
F0R5	2009-09-12 00:29	2.2	92
F0R5	2009-09-12 00:33	3.3	88
F0R5	2009-09-12 00:36	0.3	199
F0R5	2009-09-12 00:39	1.2	196

The results are:

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode
F0R5	2009-09-12 00:20	2.012	64	64	AVERAGE

Tagname	DateTime	Value	QualityDetail	OPCQuality	wwRetrievalMode
F0R5	2009-09-12 00:30	0.820	64	64	AVERAGE
F0R5	2009-09-12 00:40	1.751	64	64	AVERAGE

Quality Rule - Query 6

For RoundTrip, StateSummary, and ValueState modes, the OPC qualities are only combined with the same state in a cycle. If the state only occurs once in a cycle, then the qualities of that state will be returned. The returned NULL state will always have an OPC quality of 0 and Quality Detail of 65536. The same qualities are returned for a state that has no roundtrip in RoundTrip mode.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, StateTime
   FROM History
      WHERE TagName = 'I001'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwResolution = 10000
         AND wwRetrievalMode = 'RoundTrip'
         AND wwStateCalc = 'MaxContained'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
I001	2009-09-12 00:12	1	90
I001	2009-09-12 00:15	2	65
I001	2009-09-12 00:22	1	85
I001	2009-09-12 00:23	2	75
I001	2009-09-12 00:26	1	75
I001	2009-09-12 00:29	2	70

The results are:

Tagname	DateTime	Value	QualityDetail	OPC-Quality	StateTime
I001	2009-09-12 00:20	NULL	65536	0	NULL
I001	2009-09-12 00:20	1.0	90	90	NULL
I001	2009-09-12 00:20	2.0	65	65	NULL
I001	2009-09-12 00:20	1.0	64	64	4000
I001	2009-09-12 00:20	2.0	64	64	6000

Quality Rule - Query 7

The returned Quality Detail is the same as OPC quality unless there is special flag for certain indication; for example, when there is indication for rollover in counter mode.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality
```

```
FROM History
   WHERE TagName = 'I0R5'
      AND DateTime >= '2009-09-12 00:20'
      AND DateTime <= '2009-09-12 00:40'
      AND wwResolution = 10000
      AND wwRetrievalMode = 'Avg'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
I0R5	2009-09-12 00:07	2	218
I0R5	2009-09-12 00:14	3	218
I0R5	2009-09-12 00:22	0	218
I0R5	2009-09-12 00:25	1	218
I0R5	2009-09-12 00:27	0	218
I0R5	2009-09-12 00:29	2	218
I0R5	2009-09-12 00:33	3	218
I0R5	2009-09-12 00:36	0	218
I0R5	2009-09-12 00:39	1	218

The results are:

Tagname	DateTime	Value	QualityDetail	OPCQuality
I0R5	2009-09-12 00:20	2.6	218	218
I0R5	2009-09-12 00:30	1.0	218	218
I0R5	2009-09-12 00:40	1.6	218	218

Quality Rule - Query 8

For Interpolated mode only, the returned row with Linear wwInterpolationType will have combined qualities.

```
SELECT TagName, DateTime, Value, QualityDetail, OPCQuality, wwRetrievalMode,
wwInterpolationType
   FROM History
      WHERE TagName = 'I0R5'
         AND DateTime >= '2009-09-12 00:20'
         AND DateTime <= '2009-09-12 00:40'
         AND wwResolution = 10000
         AND wwRetrievalMode = 'Interpolated'
         AND wwInterpolationType = 'Linear'
```

If you run this query against the following sample data:

Tagname	DateTime	Resolution	QualityDetail
I0R5	2009-09-12 00:07	2	193
I0R5	2009-09-12 00:14	3	195
I0R5	2009-09-12 00:22	0	196

Tagname	DateTime	Resolution	QualityDetail
I0R5	2009-09-12 00:25	1	199
I0R5	2009-09-12 00:27	0	200
I0R5	2009-09-12 00:29	2	207
I0R5	2009-09-12 00:33	3	215
I0R5	2009-09-12 00:36	0	216
I0R5	2009-09-12 00:39	1	219

The results are:

Tagname	DateTime	Value	Quality Detail	OPC Quality	wwRetrieval Mode	wwInterpolation Type
I0R5	2009-09-12 00:20	0.8	192	192	Interpolated	Linear
I0R5	2009-09-12 00:30	2.3	192	192	Interpolated	Linear
I0R5	2009-09-12 00:40	1.0	192	219	Interpolated	Linear

Note: Cyclic, Full, Delta, Maximum, Minimum, and BestFit do not have combined qualities; therefore, the rules are not applied to these modes..

State Calculation (wwStateCalc)

The state calculation setting applies to ValueState and RoundTrip retrieval.

For ValueState retrieval, you can choose the type of state calculation (aggregation) to be performed on the data:

- **Minimum:** The shortest amount of time that the tag has been in each unique state.
- **Maximum:** The longest amount of time that the tag has been in each unique state.
- **Average:** The average amount of time that the tag has been in each unique state.
- **Total:** The total amount of time that the tag has been in each unique state.
- **Percent:** The total percentage of time that the tag has been in each unique state.
- **MinContained**: The shortest amount of time each tag has been in each unique state for each cycle, disregarding the occurrences that are not fully contained with the calculation cycle.
- **MaxContained**: The longest amount of time that the tag has been in each unique state for each cycle, disregarding the occurrences that are not fully contained with the calculation cycle.
- **AvgContained**: The average amount of time that the tag has been in each unique state for each cycle, disregarding the occurrences that are not fully contained with the calculation cycle.
- **TotalContained**: The total amount of time that the tag has been in each unique state for each cycle, disregarding the occurrences that are not fully contained with the calculation cycle.
- **PercentContained**: The percentage of time that the tag has been in each unique state for each cycle, disregarding the occurrences that are not fully contained with the calculation cycle.

All results except Percent are in milliseconds. Percent is a percentage typically between 0.0 and 100.0. The percentage can be higher than 100 in certain circumstances.

The nature of the data and how you set the cycle count determines whether you should use a "contained" version of the aggregation. The calculations apply to each unique value state that the tag was in during each retrieval cycle for the query. The total and percent calculations are always exact, but the minimum, maximum, and average calculations are subject to "arbitrary" cycle boundaries that do not coincide with the value changes. Therefore, non-intuitive results may be returned. This is most apparent for slowly-changing tags queried over long cycles.

For example, a string tag that assumes only two distinct values changing every 10 minutes is queried with a cycle time of two hours. Going into a cycle, the value (state) at the cycle boundary is recorded. If the value then changes a short while into the cycle, the state found at the cycle start time will most likely end up being the minimum value. Likewise, the state at the cycle end time is cut short at the cycle end time. The two cut-off occurrences in turn skew the average calculation.

For RoundTrip retrieval, you can only specify the following types of state calculations (aggregations) to be performed on the data. The calculations are for each unique state within each retrieval cycle for the query.

- **MinContained**. The shortest time span between consecutive leading edges of any state that occurs multiple times within the cycle, while disregarding state occurrences that are not fully contained inside of the cycle.
- **MaxContained**. The longest time span between consecutive leading edges of any state that occurs multiple times within the cycle, while disregarding state occurrences that are not fully contained inside of the cycle.
- **AvgContained**. The average time span between consecutive leading edges of any state that occurs multiple times within the cycle, while disregarding state occurrences that are not fully contained inside of the cycle. (This is the default.)
- **TotalContained**. The total time span between consecutive leading edges of any state that occurs multiple times within the cycle while disregarding state occurrences that are not fully contained inside of the cycle.
- **PercentContained**. The percentage of the cycle time spent in time span between consecutive leading edges for a state that occurs multiple times within the cycle while disregarding value occurrences that are not fully contained inside of the cycle.

Analog Value Filtering (wwFilter)

You can use the following analog filters for all retrieval modes:

- Statistical removal of outliers
- Analog to discrete conversion
- Zero around a base value

These filters are applied in a query to retrieve data from the History table, WideHistory table, or StateWideHistory table. These filter only apply to analog tags. All other types of tags, including analog summary tags, are not supported.

You need to specify a filter name in the virtual column wwFilter, with or without an override, to the set of parameters that are defined for the specified filter. The filters are specified as C-like functions: parentheses are always required, even when you choose not to override the default parameters by passing no parameters.

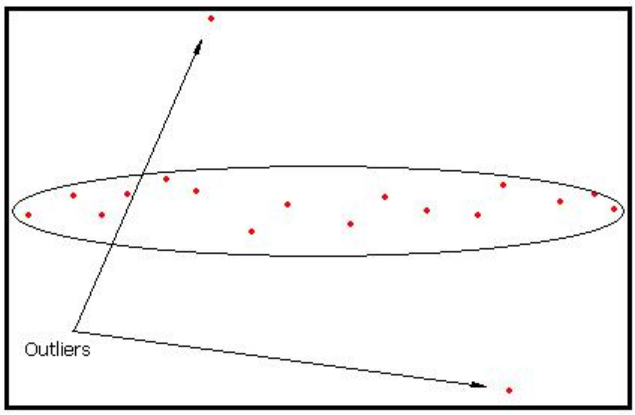
The default value for the wwFilter column is ‘NoFilter’. If the query does not specify the wwFilter element at all, or if its default value is not overridden, then no filter is applied.

When you use the analog filters in a query that uses wwQualityRule, wwQualityRule is applied first and wwFilter is applied later. You can only use one filter per query.

Statistically Removing Outliers (SigmaLimit)

This analog filter removes outliers from a set of analog points based on the assumption that the distribution of point values in the set is a normal distribution.

The following illustration shows an example of outliers.



You can filter outliers by specifying a filter called 'SigmaLimit()'. This filter has one parameter defined for specifying the value of n. This parameter is of type double. If the parameter is omitted, then a default parameter of 2.0 is used.

When this filter is specified in any retrieval mode, a time weighted mean, ì (mu), and time weighted standard deviation, ó (sigma), are found for each analog tag for the entire query range including phantom cycles if any, and points falling outside of the range [ì - nó, ì + nó] are removed from the point set before the points are processed further. In other words, the value will be filtered out if value > ì + nó or value < ì – nó.

Time weighted standard deviation is calculated as:

Math.Sqrt((integralOfSquares - 2 * timeWeightedAverage * integral + totalTime * timeWeightedAverage * timeWeightedAverage)/totalTime);

This is the single pass equivalent to the formula:

$$\sigma^2_{\text{weighted}} = \sum_{i=1}^{N} w_i \left(x_i - \mu^*\right)^2$$

Ranges where the value is NULL are excluded from these calculations.

A cyclic query example using a 'SigmaLimit()' filter without specifying the n value would look like this:

```
SELECT DateTime, Value, wwFilter
   FROM History
      WHERE TagName = ('TankLevel')
         AND DateTime >= '2008-01-15 15:00:00'
         AND DateTime <= '2008-01-15 17:00:00'
         AND wwRetrievalMode = 'Cyclic'
         AND wwFilter = 'SigmaLimit()'
```

Not specifying the n-value as done here is the same as specifying 'SigmaLimit(2)'. The result set might look like this:

DateTime	Value	wwFilter
2008-01-15 15:00:00.000	34.56	SigmaLimit()
2008-01-15 16:00:00.000	78.90	SigmaLimit()
2008-01-15 17:00:00.000	12.34	SigmaLimit()

If the first value would be filtered out by the SigmaLimit filter, the value will be replaced with the time weighted mean.

Converting Analog Values to Discrete Values (ToDiscrete)

The analog to discrete conversion filter allows you to convert value streams for any analog tag in the query tag list into discrete value streams. The filter can be used with all the retrieval modes.

To convert analog values to discrete values, specify the ToDiscrete() filter in the wwFilter column. This filter has two parameters:

Parameter	Valid Values	Default Value
CutoffValue	any double value	0.0
Operator	>, >=, or <=	>

The following are supported syntaxes.

- ToDiscrete()
- ToDiscrete(2)
- ToDiscrete(2, >=)

The following are unsupported syntaxes.

- ToDiscrete(2,)
- ToDiscrete(, >=)
- ToDiscrete(>=)

The cutoff value holds the value that signifies the boundary between values that are to be interpreted as OFF and values that are to be interpreted as ON.

The operator parameter controls the value range relative to the cutoff value to convert to the ON value and vice versa.

NULLs encountered in the analog value stream are copied unchanged to the discrete value stream. The quality of each discrete point is copied from the analog point that causes its production. However, the quality detail for values modified with this filter will have the QualityDetail flag 0x2000 (value changed by filter) set. For example, consider the following ValueState query:

```
SELECT DateTime, vValue, StateTime, wwFilter
   FROM History
      WHERE TagName IN ('SysTimeSec')
         AND DateTime >= '2008-01-15 15:00:00'
         AND DateTime <= '2008-01-15 17:00:00'
         AND wwRetrievalMode = 'ValueState'
```

```
        AND wwStateCalc = 'MinContained'
        AND wwResolution = 7200000
        AND wwFilter = 'ToDiscrete(29, >)'
```

Here the operator is specified as >, so values greater than but not including 29 are internally converted to ON, whereas values from 0 to 29 are converted to OFF. This query could return the following rows:

DateTime	vValue	StateTime	wwFilter
2008-01-15 15:00:00.000	0	30000	ToDiscrete(29, >)
2008-01-15 15:00:00.000	1	30000	ToDiscrete(29, >)
2008-01-15 17:00:00.000	0	30000	ToDiscrete(29, >)
2008-01-15 17:00:00.000	1	30000	ToDiscrete(29, >)

The values returned in the StateTime column show that the shortest amount of time that SysTimeSec had values equivalent to either ON or OFF and remained in that state was 30 seconds. A similar RoundTrip query would look like this:

```
SELECT DateTime, vValue, StateTime, wwFilter
   FROM History
      WHERE TagName IN ('SysTimeSec')
         AND DateTime >= '2008-01-15 15:00:00'
         AND DateTime <= '2008-01-15 17:00:00'
         AND wwRetrievalMode = 'RoundTrip'
         AND wwStateCalc = 'MaxContained'
         AND wwResolution = 7200000
         AND wwFilter = 'ToDiscrete(29, <=)'
```

Here the operator is specified as <=, so the resulting conversion is exactly opposite to that performed in the previous query. Now values smaller than or equal to 29 are internally converted to ON, whereas values from 30 to 59 are converted to OFF. This query could return the following rows:

DateTime	vValue	StateTime	wwFilter
2008-01-15 15:00:00.000	0	60000	ToDiscrete(29, <=)
2008-01-15 15:00:00.000	1	60000	ToDiscrete(29, <=)
2008-01-15 17:00:00.000	0	60000	ToDiscrete(29, <=)
2008-01-15 17:00:00.000	1	60000	ToDiscrete(29, <=)

The values returned in the StateTime column now show that the longest amount of time found between roundtrips for both the OFF and the ON state within the 2-hour cycles was 60 seconds.

Using the ToDiscrete() filter is similar to using edge detection for event tags. Edge detection returns the actual value with a timestamp in history for when a value matched a certain criteria. The ToDiscrete() filter returns either a 1 or 0 to show that the criteria threshold was crossed. The ToDiscrete() filter is more flexible, however, in the following ways:

- You can use it with delta and full retrieval.
- You can combine it with "time-in-state" calculations to determine how long a value is above a certain threshold or the duration between threshold times.

Use the ToDiscrete() filter if you are mostly interested in when something occurred, and not necessarily the exact value of the event.

For more information on edge detection, see *Edge Detection for Events (wwEdgeDetection)* on page 78.

"Zeroing" Around a Base Value (SnapTo)

This analog filter lets you force values in a well-defined range around one or more base values to "snap to" that base value. For example, you can use this filter when a tank is known to be empty, but the tag that stores the tank level returns a "noisy" value close to zero.

The filter can be used with all retrieval modes, but its main benefits are in the aggregate retrieval modes: average, integral, minimum, and maximum.

To zero values around the base value, specify the SnapTo() filter in the wwFilter column of the query.

The syntax for this filter is:

```
SnapTo([tolerance[,base_value_1[, base_value_2]…]])
```

This filter has two parameters:

Parameter	Valid Values	Default Value
Tolerance	any double value	0.01
BaseValue	zero, one, or up to 100 comma-separated double values	single base value of 0.0

The following are supported syntaxes.

- SnapTo() – Same as SnapTo(0.01, 0.0)
- SnapTo(3.7) – Same as SnapTo(3.7, 0.0)
- SnapTo(3,) – Syntax Error
- SnapTo(,0) – Syntax error
- SnapTo(,) – Syntax error
- SnapTo(3, 4, -5) – Tolerance=3, Base Values 4 and -5.

When the Snap to filter is specified, point values falling inside any of the ranges [Base value – Tolerance, Base value + Tolerance] will be forced to the base value before the point goes into further retrieval processing. The result is undefined if the base value +/- tolerance exceeds the range of the double data type. The range is calculated using this expression:

```
If (x <= Base value + Tolerance AND x >= Base value – Tolerance)
x = Base value
```

where *x* is the value of the point then if ranges overlap, the first matching base value will be used.

A query example from the History table looks like this:

```
SELECT DateTime, Value, wwFilter
   FROM History
      WHERE TagName = ('TankLevel')
         AND DateTime >= '2008-01-15 15:00:00'
         AND DateTime <= '2008-01-15 17:00:00'
```

```
AND wwRetrievalMode = 'Average'
AND wwResolution = 3600000
AND wwFilter = 'SnapTo(0.01, 0, 1000)'
```

The following rows might be returned:

DateTime	Value	wwFilter
2008-01-15 15:00:00.000	0	SnapTo(0.01, 0, 1000)
2008-01-15 16:00:00.000	875.66	SnapTo(0.01, 0, 1000)
2008-01-15 17:00:00.000	502.3	SnapTo(0.01, 0, 1000)

When a value is snapped, the QualityDetail bit flag 0x2000 is set.

If the filter syntax is not correct, a syntax error is returned and no rows are returned.

Selecting Values for Analog Summary Tags (wwValueSelector)

For an analog summary tag, multiple summarized values can be stored in the historian for a single summarization period. When you query analog summary data, a single value, time, and quality (VTQ) must first be extrapolated from the summarized values.

You set the value selector in the query to specify which summarized value to return. The possible values are as follows:

Value Selector Setting	Value Returned	Timestamp Returned
AUTO	The retrieval mode determines the value. See the following table for how AUTO applies to the value selection. This is the default value.	The retrieval mode determines the timestamp. See the following table for how AUTO applies to the value selection. This is the default value.
FIRST	The first value that occurs within the summary period.	The actual timestamp of the first value occurrence within the summary period.
LAST	The last value that occurs within the summary period.	The actual timestamp of the last value occurrence within the summary period.
MIN or MINIMUM	The first minimum value that occurs within the summary period.	The actual timestamp of the first minimum value occurrence within the summary period.
MAX or MAXIMUM	The first maximum value that occurs within the summary period.	The actual timestamp of the first maximum value occurrence within the summary period.
AVG or AVERAGE	A time-weighted average calculated from values within the summary period.	The summary period start time.

Value Selector Setting	Value Returned	Timestamp Returned
INTEGRAL	An integral value calculated from values within the summary period.	The summary period start time.
STDDEV or STANDARDDEVIATION	A standard deviation calculated from values within the summary period.	The summary period start time.

The following table describes the value to be considered if the value selector is set to AUTO:

Retrieval Mode	Analog Summary Behavior
Cyclic	The last value within the summary period is used. The actual timestamp of the last value occurrence within the summary period is used.
Delta	The last value within the summary period is used. The actual timestamp of the last value occurrence within the summary period is used.
Full	The last value within the summary period is used. The actual timestamp of the last value occurrence within the summary period is used.
Interpolated	The retrieval mode determines the appropriate value to return. See the following table for how AUTO applies to the value selection. This is the default value.
Best Fit	The first, last, min, and max points from analog summaries are all considered as analog input points. Best fit analysis is done with these points. If the analog summary percentage good is not 100%, the cycle is considered to have a NULL.
Average	The averages of analog summaries are calculated using the values from the Average column of the AnalogSummaryHistory table. Interpolation type is ignored for analog summary values, and STAIRSTEP interpolation is always used. PercentGood is calculated by considering the TimeGood of each analog summary. If cycle boundaries do not exactly match the summary periods of the stored analog summaries, the averages and time good are calculated by prorating the average and time good values for the portion of the time the summary period overlaps with the cycle. Quality will be set to 64 (uncertain) if cycle boundaries do not match summary periods. If the QualityDetail of any analog summary considered for a cycle is uncertain (64), the resulting quality is set to 64.
Minimum	The first minimum value within the summary period is used. The actual timestamp of the first minimum value occurrence within the summary period is used.
Maximum	The first maximum value within the summary period is used. The actual timestamp of the first maximum value occurrence within the summary period is used.

Retrieval Mode	Analog Summary Behavior
Integral	The integrals of analog summaries are calculated using the Integral column of the AnalogSummaryHistory table. Interpolation type is ignored for analog summary values, and STAIRSTEP interpolation is always used. PercentGood is calculated by considering the TimeGood of each analog summary. If cycle boundaries do not exactly match the summary periods of the stored analog summaries, the integrals and time good are calculated by prorating the integral and time good values for the portion of the time the summary period overlaps with the cycle. Quality is set to 64 (uncertain) if cycle boundaries do not match summary periods. If the QualityDetail of any analog summary considered for a cycle is uncertain (64), the resulting quality will be set to 64.
Slope	The last value within the summary period is used. The actual timestamp of the last value occurrence within the summary period is used.
ValueState	Cannot be used with analog summary data. No values are returned.
Counter	Cannot be used with analog summary data. No values are returned.
RoundTrip	Cannot be used with analog summary data. No values are returned.

For an analog summary tag, if any of the data within a summary period has an OPCQuality other than Good, the OPCQuality returned will be Uncertain. This is true even for summary values that are not calculated, such as first, last, minimum, maximum, and so on. For example, if the OPCQuality for a last value is actually Good, but there was a I/O Server disconnect during the summary calculation period, the OPCQuality for the last value is returned as Uncertain. A QualityDetail of 202 is used to distinguish between the original point and the latest point.

CHAPTER 3

SQL Query Examples

In addition to query examples that use the AVEVA Historian time domain extensions, other query examples are provided to demonstrate how to perform more complex queries or to further explain how retrieval works.

The examples provided are not exhaustive of all possible database queries, but they should give you an idea of the kinds of queries that you could write.

For general information on creating SQL queries, see your Microsoft SQL Server documentation.

Note: If you have configured SQL Server to be case-sensitive, be sure that you use the correct case when writing queries.

Querying the History Table

The History table presents acquired plant data in a historical format. For more information, see History Tables and Views in the *AVEVA Historian Database Reference*.

The following query returns the date/time stamp and value for the tag "ReactLevel." The query uses the remote table view (History is used in place of INSQL.Runtime.dbo.History).

If you do not specify a wwCycleCount or wwResolution, the query will return 100 rows (the default).

```
SELECT DateTime, Sec = DATEPART(ss, DateTime), TagName, Value
   FROM History
      WHERE TagName = 'ReactLevel'
         AND DateTime >= '2001-03-13 1:15:00pm'
         AND DateTime <= '2001-03-13 2:15:00pm'
         AND wwRetrievalMode = 'Cyclic'
```

The results are:

DateTime	Sec	TagName	Value
2001-03-13 13:15:00.000	0	ReactLevel	1775.0
2001-03-13 13:15:00.000	36	ReactLevel	1260.0
2001-03-13 13:16:00.000	12	ReactLevel	1650.0
2001-03-13 13:16:00.000	49	ReactLevel	1280.0
2001-03-13 13:17:00.000	25	ReactLevel	1525.0
2001-03-13 13:18:00.000	1	ReactLevel	585.0
2001-03-13 13:18:00.000	38	ReactLevel	1400.0
2001-03-13 13:19:00.000	14	ReactLevel	650.0
2001-03-13 13:19:00.000	50	ReactLevel	2025.0
2001-03-13 13:20:00.000	27	ReactLevel	765.0
2001-03-13 13:21:00.000	3	ReactLevel	2000.0

2001-03-13 13:21:00.000	39	ReactLevel	830.0
2001-03-13 13:22:00.000	16	ReactLevel	1925.0
...			
(100 row(s) affected)			

Querying the Live Table

The Live table presents the latest available streamed data for each tag in the table.

Note: In certain situations, data can bypass the Live table. These situations include:
- Receiving non-streamed original data (store/forward or CSV);
- Receiving revision data for a Latest value;
- Receiving no new streamed values after Historian was shut down and disabled, or after the computer was rebooted.

For more information, see History Tables and Views in the *AVEVA Historian Database Reference*.

The following query returns the current value of the specified tag. The query uses the remote table view (Live is used in place of INSQL.Runtime.dbo.Live).

```
SELECT TagName, Value
   FROM Live
      WHERE TagName = 'ReactLevel'
```

The result is:

TagName	Value
ReactLevel	1145.0
(1 row(s) affected)	

Querying the WideHistory Table

The wide extension table is a transposition of the History table. Use the wide history tables any time you want to find the value of one or more tags over time and need to specify different filter criteria for each tag.

For more information, see History Tables and Views in the *AVEVA Historian Database Reference*.

The following query returns the value of two tags from the WideHistory table. The WideHistory table can only be accessed using the OPENQUERY function. The "Runtime.dbo." qualifier is optional.

```
SELECT * FROM OpenQuery(INSQL,'
   SELECT DateTime, ReactLevel, ReactTemp
   FROM Runtime.dbo.WideHistory
      WHERE Reactlevel > 1500
      AND ReactTemp > 150
   ')
```

The results are:

DateTime	ReactLevel	ReactTemp

2001-03-02 06:20:00.000	1865.0	191.3
2001-03-02 06:21:00.000	2025.0	195.9
2001-03-02 06:22:00.000	2000.0	195.9
2001-03-02 06:23:00.000	2025.0	180.9
2001-03-02 06:27:00.000	1505.0	177.5
(5 row(s) affected).		

In the WideHistory table, the column type is determined by the tag type.

```
SELECT * FROM OpenQuery(INSQL, 'SELECT DateTime, SysTimeMin, SysPulse, SysString
FROM WideHistory
      WHERE DateTime >= "2001-12-20 0:00"
          AND DateTime <= "2001-12-20 0:05"
          AND wwRetrievalMode = "delta"
      ')
```

The results are:

DateTime	SysTime Min	SysPulse	SysString
2001-12-20 00:00:00.000	0	0	2001/12/20 08:00:00
2001-12-20 00:01:00.000	1	1	2001/12/20 08:00:00
2001-12-20 00:02:00.000	2	0	2001/12/20 08:00:00
2001-12-20 00:03:00.000	3	1	2001/12/20 08:00:00
2001-12-20 00:04:00.000	4	0	2001/12/20 08:00:00
2001-12-20 00:05:00.000	5	1	2001/12/20 08:00:00

Querying Wide Tables in Delta Retrieval Mode

Wide tables in delta retrieval mode will behave normally if only one tag is returned. However, for a multiple tag display, a complete row is returned to the client for each instance in which one or more of the tags in the query returns a different value. The row will reflect the actual values being returned for the tags returning results, and will reflect the previous values for the remaining tags in the result set (similar to cyclic retrieval).

Note: The value can be "invalid" or some other quality value.

The following query returns values for three tags from the WideHistory table. "MyTagName" is a tag that periodically is invalid.

```
SELECT * FROM OpenQuery(INSQL,'
   SELECT DateTime, SysTimeSec, SysTimeMin, MyTagName
      FROM WideHistory
          WHERE DateTime >= "2001-05-12 13:00:00"
          AND DateTime <= "2001-05-12 13:02:00"
          AND wwRetrievalMode = "Delta"
   ')
```

The results are:

DateTime	SysTimeSec	SysTimeMin	MyTagName
...			
2001-05-12 13:00:55.000	55	00	1
2001-05-12 13:00:56.000	56	00	1
2001-05-12 13:00:57.000	57	00	1
2001-05-12 13:00:57.500	57	00	null
2001-05-12 13:00:58.000	58	00	null
2001-05-12 13:00:59.000	59	00	null
2001-05-12 13:01:00.000	00	01	null
2001-05-12 13:01:00.500	00	01	2
2001-05-12 13:01:01.000	01	01	2
2001-05-12 13:01:02.000	02	01	2
2001-05-12 13:01:03.000	03	01	2
...			

Notice that 57 appears twice since the occurrence of 1 changing to NULL for tag "MyTagName" occurs sometime between the 57th and 58th second. The same applies for NULL changing to 2. The same behavior applies to discrete values.

Querying the AnalogSummaryHistory View

The AnalogSummaryHistory view is a "wide" view that allows you to return multiple statistics for a single tag within a single query.

The following query returns the minimum, maximum, and average values for the SysTimeSec tag for the last hour.

```
declare @End datetime
set @End = left(convert(varchar(30),getdate(),120),14)+'00:00'
SELECT Tagname, OPCQuality, Minimum as MIN, Maximum as MAX, Average as AVG
   FROM AnalogSummaryHistory
      WHERE TagName = 'SysTimeSec'
         AND StartDateTime >= dateadd(minute,-60,@End)
         AND EndDateTime < @End
         AND wwCycleCount = 2
```

The results are:

Tagname	OPCQuality	MIN	MAX	AVG
SysTimeSec	192	0	59	29.5

Note: When querying the AnalogSummaryHistory view, a data point occuring at the same timestamp as the `EndDateTime` is not considered part of the query interval. Instead, it is treated as the first data point in the next interval beginning at `EndDateTime`, and so is not included in the maximum or average value calculations.

Querying the StateSummaryHistory View

The StateSummaryHistory view is a "wide" view that allows you to return multiple statistics for a single tag within a single query.

The following query returns the state count, total time in state, and the percentage of time in state for the SysPulse system tag for the last hour. One row is returned for each state.

```
DECLARE @End DateTime
SET @END = getdate()
   SELECT
      TagName,
      Value,
      OPCQuality,
      StateCount,
      StateTimeTotal,
      StateTimePercent
   FROM dbo.StateSummaryHistory
   WHERE TagName = N'SysPulse'
      AND StartDateTime >= dateadd(minute,-60,@End)
      AND EndDateTime <= @End
      AND wwCycleCount = 1
```

The results are:

TagName	Value	OPCQuality	State Count	StateTimeTotal	StateTimePercent
SysPulse	1	192	30	1800000	50
SysPulse	0	192	30	1800000	50

The following query returns the minimum time in state, the minimum contained time in state, and value for the SysTimeSec system tag.

```
SELECT
   TagName,
   StartDateTime,
   EndDateTime,
   StateTimeMin as STM,
   StateTimeMinContained as STMC,
   Value
FROM StateSummaryHistory
WHERE TagName='SysTimeSec'
   AND wwRetrievalMode='Cyclic'
   AND wwResolution=5000
   AND StartDateTime>='2009-10-21 17:40:00.123'
   AND StartDateTime<='2009-10-21 17:40:05.000'
```

The results are:

TagName	StartDateTime	EndDateTime	STM	STMC	Value
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	877	0	0
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	1000	1000	1

TagName	StartDateTime	EndDateTime	STM	STMC	Value
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	1000	1000	2
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	1000	1000	3
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	1000	1000	4
SysTimeSec	2009-10-21 17:40:00.123	2009-10-21 17:40:05.123	123	0	5

Using SliceBy

You can retrieve summary statistics for a tag per batch by using the **SliceBy** parameter in a query. Batches can be defined by changes in a different tag, such as a batch ID or valve position.

For example, suppose you wanted statistics on flow rate depending on the position of a valve. Each time the valve position changes, a new batch is reported. Here is a query to retrieve that information:

```
select SliceByValue, TagName, StartDateTime, EndDateTime, OPCQuality,
PercentGood, wwResolution, Average
from AnalogSummaryHistory
where TagName='M31.FlowIn'
and SliceBy='M31.ValveIn'
and StartDateTime>='2018-11-27 0:00'
and EndDateTime<='2018-11-28 0:00'
```

The results are:

SliceBy Value	TagName	StartDateTime	EndDateTime	OPCQuality	PercentGood	wwResolution	Average
0	M31.FlowIn	2018-11-27 07:56:11	2018-11-27 09:49:47	192	100	6816000	57.4379
1	M31.FlowIn	2018-11-27 09:49:47	2018-11-27 15:23:23	192	100	20016000	0.5108
0	M31.FlowIn	2018-11-27 15:23:23	2018-11-27 17:24:17	192	100	7254000	50.3615
1	M31.FlowIn	2018-11-27 17:24:17	2018-11-27 21:50:59	192	100	16002000	0.5220
0	M31.FlowIn	2018-11-27 21:50:59	2018-11-27 23:47:17	192	100	6975000	51.4363

This query uses the same data. This time, using SliceByValue, the query retrieves statistics on the flow rate, but only for those batches when the valve is open.

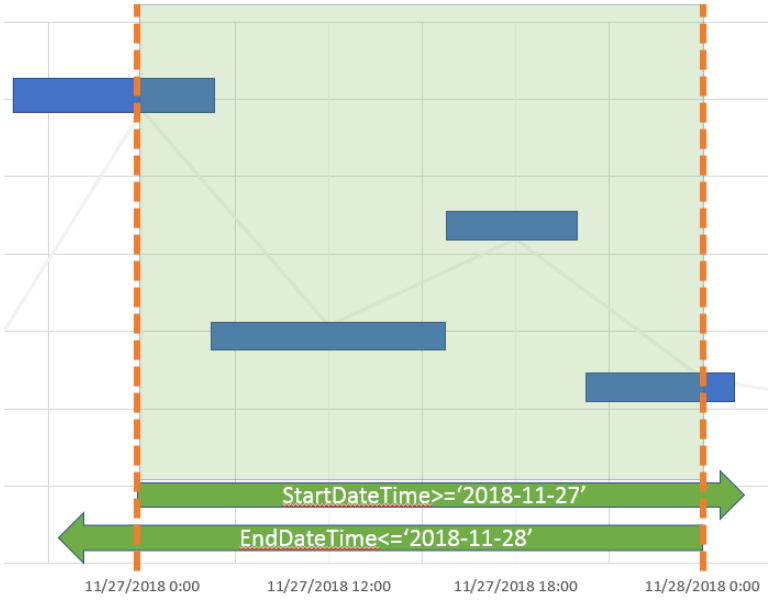
```
select SliceByValue, TagName, StartDateTime, EndDateTime, OPCQuality,
PercentGood, wwResolution, Average
from AnalogSummaryHistory
```

```
where TagName='M31.FlowIn'
and SliceBy='M31.ValveIn'
and SliceByValue=1
and StartDateTime>='2018-11-27 0:00'
and EndDateTime<='2018-11-28 0:00'
```

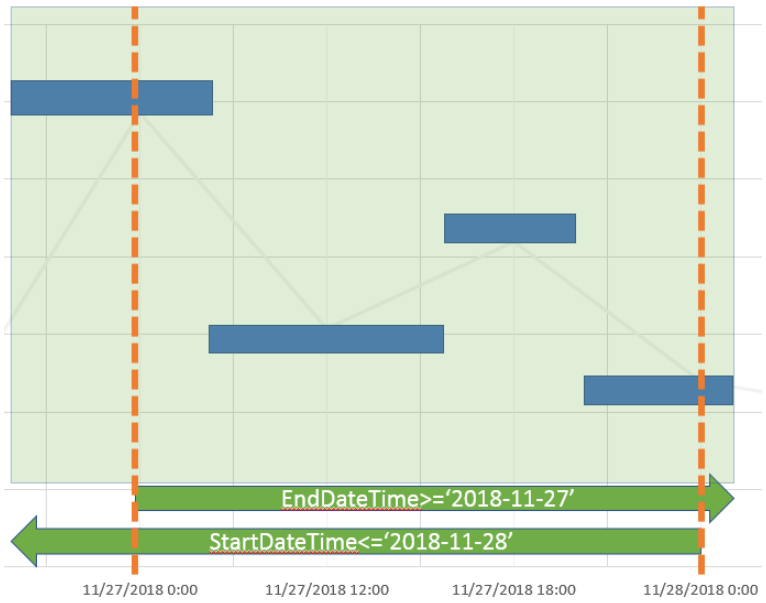
The results this time are:

SliceBy Value	TagName	StartDateTime	EndDateTime	OPCQuality	Percent Good	wwResolution	Average
1	M31.FlowIn	2018-11-27 09:49:47	2018-11-27 15:23:23	192	100	20016000	0.5108
1	M31.FlowIn	2018-11-27 17:24:17	2018-11-27 21:50:59	192	100	16002000	0.5220

In the previous two queries, "StartDateTime>=" and "EndDateTime<=" define boundaries for the query to include only those batches that both start and end within the boundaries set. Any batches that begin before or end after the reporting period are excluded.



This next query is the same as the previous one, but swaps the start and end time criteria to include batches that begin before or end after the reporting period:




```
select SliceByValue, TagName, StartDateTime, EndDateTime, OPCQuality,
PercentGood, wwResolution, Average
from AnalogSummaryHistory
where TagName='M31.FlowIn'
and SliceBy='M31.ValveIn'
and SliceByValue=1
and EndDateTime>='2018-11-27 0:00'
and StartDateTime<='2018-11-28 0:00'
```

The results are:

SliceByValue	TagName	StartDateTime	EndDateTime	OPCQuality	PercentGood	wwResolution	Average
1	M31.FlowIn	2018-11-26 23:08:08	2018-11-27 07:56:11	192	100	31683000	0.5386
1	M31.FlowIn	2018-11-27 09:49:47	2018-11-27 15:23:23	192	100	20016000	0.5108
1	M31.FlowIn	2018-11-27 17:24:17	2018-11-27 21:50:59	192	100	16002000	0.5220
1	M31.FlowIn	2018-11-27 23:47:14	2018-11-28 10:53:20	192	100	39966000	0.5039

SliceBy Queries That Include Future

If you define a timeframe that includes a batch that is not completed, you will get an summary statictics for the batch data currently available, but it will be marked with "OPCQuality=64", meaning the quality is uncertain because the end time is unknown. An ending batch can be incomplete because:

- The batch is currently in progress and the next batch has not yet initiated. Batches are defined by changes in a different tag. If that tag has not yet changed, the current batch is in progress and the next batch has not initiated.
- The batch did physically completed, but the data hasn't yet been written to the server due to latency, power-outage, etc.

Here is an example. This query retrieves average values for batches that completed on Nov 28. The "select getdate()" line shows that the current timestamp for the query is Nov 28, 2018 at 4:31. Since the day is not over, the system cannot know for sure that another batch won't be initiated. Therefore, the last retrieved batch is incomplete.

```
select SliceByValue, TagName, StartDateTime, EndDateTime, OPCQuality,
PercentGood, wwResolution, Average
from AnalogSummaryHistory
where TagName='M31.Level'
and SliceBy='M31.Batch'
and EndDateTime>='2018-11-28 0:00'
and StartDateTime<='2018-11-29 0:00'
select getdate()
```

The results are:

SliceByValue	TagName	StartDateTime	EndDateTime	OPCQuality	PercentGood	wwResolution	Average
CRC-4	M31.Level	2018-11-27 21:44:23	2018-11-28 03:03:41	192	100	19158000	94.9624
	M31.Level	2018-11-28 03:03:41	2018-11-28 04:12:20	192	100	4119000	1.6700
BLB-7	M31.Level	2018-11-28 04:12:20	2018-11-28 00:00:00	64	1.7020	NULL	134.8897

(No column name)
2018-11-28 04:31:33

Using Group By with SliceBy

You can use **Group By** in a query to get an overall summary off all occurrences of a state. For example, you might want an overall summary of production when a piece of equipment was in a certain position.

Here is an example of using Group By to organize data by tag state. This query retrieves an overall summary of statistics for each position of the valve. One line summarizes all batches for the day where the valve was closed (0) and another line for all batches with the valve open (1).

```
select SliceByValue, TagName, StartDateTime=min(StartDateTime),
EndDateTime=max(EndDateTime), TotalTime=sum(wwResolution),
Maxiumum=max(Maximum), Total=sum(Integral),
Average=sum(Average*wwResolution)/sum(wwResolution),
AvgOfAvg=avg(Average)
from AnalogSummaryHistory
where TagName='M31.FlowIn'
and SliceBy='M31.ValveIn'
and EndDateTime>='2018-11-27 0:00'
and StartDateTime<='2018-11-28 0:00'
group by TagName, SliceByValue
```

The results are:

TagName	SliceByValue	StartTime	EndTime	TotalTime	Maximum	Total	Average	AvgOfAvg
M31.FlowIn	0	2018-11-27 09:49:47	2018-11-27 21:50:59	36018000	6.75	309.66	0.51	0.51
M31.FlowIn	1	2018-11-27 07:56:11	2018-11-27 23:47:14	21045000	114.92	18593.13	53.00	53.078

Note: This example correctly calculates the overall average for each state in the "Average" column by weighting the duration of each state. As explained by Simpson's Paradox, the simpler, "AvgOfAvg" calculation is not statistically accurate and can differ significantly with some data sets.

Using an Unconventional Tagname in a Wide Table Query

In a SQL query against a wide table, unconventional tag names must be delimited with brackets ([ ]), because the tagname is used as a column name. For example, tagnames containing a minus (-) or a forward slash (/) must be delimited, otherwise the parser will attempt to perform the corresponding arithmetic operation. No error will result from using brackets where not strictly necessary. For more information on unconventional tagnames, see Tag Naming Conventions.

The following is an example of how to delimit a tagname in a query on a wide table. "ReactTemp-2" and "ReactTemp+2" are tagnames. Without the delimiters, the parser would attempt to include the "-2" and "+2" suffixes on the tagnames as part of the arithmetic operation.

For clarity and maintainability of your queries, however, it is recommended that you do not use special characters in tagnames unless strictly necessary.

```
SELECT * FROM OpenQuery(INSQL,
    'SELECT ReactTemp, [ReactTemp-2]-2, [ReactTemp+2]+2 FROM WideHistory WHERE
    ... ')
```

Using an INNER REMOTE JOIN

Instead of using " … `WHERE TagName IN (SELECT TagName …)` ", it is more efficient to use INNER REMOTE JOIN syntax.

In general, use the following pattern for INNER REMOTE JOIN queries against the historian:

<SQLServerTable> INNER REMOTE JOIN <HistorianExtensionTable>

This query returns data from the history table, based on a string tag that you filter for from the StringTag table:

```
SELECT DateTime, T.TagName, vValue, Quality, QualityDetail
    FROM StringTag T inner remote join History H
    ON T.TagName = H.TagName
        WHERE T.MaxLength = 64
            AND DateTime >='2002-03-10 12:00:00.000'
            AND DateTime <='2002-03-10 16:40:00.000'
            AND wwRetrievalMode = 'Delta'
```

This query returns data from the history table, based on a discrete tag that you filter for from the Tag table:

```
SELECT DateTime, T.TagName, vValue, Quality, QualityDetail
    FROM Tag T inner remote join History H
    ON T.TagName = H.TagName
        WHERE T.TagType = 2
            AND T.Description like 'Discrete%'
            AND DateTime >='2002-03-10 12:00:00.000'
            AND DateTime <='2002-03-10 16:40:00.000'
            AND wwRetrievalMode = 'Delta'
```

Setting Both a Time and Value Deadband for Retrieval

If both time and value deadbands are specified, then every sample is checked for both deadbands, against the current basis value (the last sample returned).

If it passes both tests, then it is returned and acts as the basis for checking the next sample.

For example:

```
SELECT DateTime, TagName, Value
    FROM History
        WHERE TagName = 'ReactTemp'
            AND DateTime >= '2002-03-13 10:08'
            AND DateTime <= '2002-03-13 10:28'
            AND wwRetrievalMode = 'Delta'
            AND wwTimeDeadband = 5000
            AND wwValueDeadband = 5
```

The tag selected, ReactTemp, has a MinEU value of 0 and a MaxEU value of 220. Thus, the value deadband will be 5 percent of (220 - 0), which equals 11. ReactTemp changes rapidly between its extreme values, but the value remains constant for short periods near the high and low temperature limits. Therefore, when changes are rapid, the value deadband condition is satisfied first, then the time deadband is satisfied. In this region, the behavior is dominated by the time deadband, and the returned rows are spaced at 5 second intervals. Where the temperature is more constant (particularly at the low temperature end), the time deadband is satisfied first, followed by the value deadband. Both deadbands are satisfied only when the value of a row is more than 11 degrees different from the previous row. Thus, the effect of value deadband can be seen to dominate near the low and high temperature extremes of the tag.

The results are:

DateTime	TagName	Value
2002-03-13 10:08:00.000	ReactTemp	121.0
2002-03-13 10:08:10.000	ReactTemp	189.10000610351562
2002-03-13 10:08:20.000	ReactTemp	147.69999694824219
2002-03-13 10:08:30.000	ReactTemp	106.30000305175781
2002-03-13 10:08:40.000	ReactTemp	30.100000381469727
2002-03-13 10:08:50.000	ReactTemp	16.399999618530273
2002-03-13 10:09:00.000	ReactTemp	61.0
2002-03-13 10:09:10.000	ReactTemp	151.0
2002-03-13 10:09:20.000	ReactTemp	173.0
2002-03-13 10:09:30.000	ReactTemp	131.60000610351562
2002-03-13 10:09:40.000	ReactTemp	57.700000762939453
2002-03-13 10:09:50.000	ReactTemp	16.299999237060547
2002-03-13 10:10:10.000	ReactTemp	96.0
2002-03-13 10:10:20.000	ReactTemp	186.0
2002-03-13 10:10:30.000	ReactTemp	156.89999389648437
2002-03-13 10:10:40.000	ReactTemp	115.5
2002-03-13 10:10:50.000	ReactTemp	41.599998474121094
2002-03-13 10:11:00.000	ReactTemp	21.0
2002-03-13 10:11:10.000	ReactTemp	41.0
2002-03-13 10:11:20.000	ReactTemp	131.0
2002-03-13 10:11:30.000	ReactTemp	184.5
2002-03-13 10:11:40.000	ReactTemp	140.80000305175781
2002-03-13 10:11:50.000	ReactTemp	99.400001525878906
2002-03-13 10:12:00.000	ReactTemp	25.5
2002-03-13 10:12:20.000	ReactTemp	76.0
2002-03-13 10:12:30.000	ReactTemp	166.0
2002-03-13 10:12:50.000	ReactTemp	124.69999694824219

DateTime	TagName	Value
2002-03-13 10:13:00.000	ReactTemp	50.799999237060547
2002-03-13 10:13:10.000	ReactTemp	16.399999618530273
2002-03-13 10:13:30.000	ReactTemp	111.0
2002-03-13 10:13:40.000	ReactTemp	193.69999694824219
2002-03-13 10:13:50.000	ReactTemp	152.30000305175781
2002-03-13 10:14:00.000	ReactTemp	108.59999847412109
2002-03-13 10:14:10.000	ReactTemp	34.700000762939453
2002-03-13 10:14:20.000	ReactTemp	21.0
2002-03-13 10:14:30.000	ReactTemp	51.0
2002-03-13 10:14:40.000	ReactTemp	146.0
2002-03-13 10:14:50.000	ReactTemp	177.60000610351562
2002-03-13 10:15:00.000	ReactTemp	136.19999694824219
2002-03-13 10:15:10.000	ReactTemp	92.5
2002-03-13 10:15:20.000	ReactTemp	18.600000381469727
2002-03-13 10:15:40.000	ReactTemp	86.0
2002-03-13 10:15:50.000	ReactTemp	181.0
2002-03-13 10:16:00.000	ReactTemp	161.5
2002-03-13 10:16:10.000	ReactTemp	120.09999847412109
2002-03-13 10:16:20.000	ReactTemp	76.400001525878906
2002-03-13 10:16:30.000	ReactTemp	20.899999618530273
2002-03-13 10:16:50.000	ReactTemp	81.0
2002-03-13 10:17:00.000	ReactTemp	176.0
2002-03-13 10:17:10.000	ReactTemp	163.80000305175781
2002-03-13 10:17:20.000	ReactTemp	122.40000152587891
2002-03-13 10:17:30.000	ReactTemp	46.200000762939453
2002-03-13 10:17:40.000	ReactTemp	18.700000762939453
2002-03-13 10:18:00.000	ReactTemp	116.0
2002-03-13 10:18:10.000	ReactTemp	189.10000610351562
2002-03-13 10:18:20.000	ReactTemp	147.69999694824219
...		

Using wwResolution, wwCycleCount, and wwRetrievalMode in the Same Query

The results of a database query will vary depending on the combination of resolution, cycle count, and retrieval mode that you use in the query. These results are summarized in the following table (where *N* is a numeric value):

Retrieval Mode	Resolution	Cycle Count	Results
CYCLIC	*N*	0 (or no value)	All stored data for tags during the specified time interval are queried, and then a resolution of N ms applied.
CYCLIC	0 (or no value)	0	The server will return 100,000 rows per tag specified.
CYCLIC	0 (or no value)	*N*	All stored data for tags during the specified time interval are queried, and then a cycle count of N evenly spaced rows is applied.
CYCLIC	*N*	(any value is ignored)	All stored data for tags during the specified time interval are queried, and then a resolution of N ms applied.
CYCLIC	(no value)	(no value or a value less than 0)	The server will return 100 rows per tag specified.
DELTA	(any value is ignored)	0	All values that changed during the specified time interval are returned (up to 100,000 rows total).
DELTA	(any value is ignored)	*N*	Values that changed during the specified time interval are queried, and then a cycle count (first N rows) is applied. The cycle count limits the maximum number of rows returned, regardless of how many tags were queried. For example, a query that applies a cycle count of 20 to four tags will return a maximum of 20 rows of data. An initial row will be returned for each tag, and the remaining 16 rows will be based on subsequent value changes for any tag.
DELTA	(any value is ignored)	(no value)	All values that changed during the specified time interval are returned (no row limit).

In general, if there is an error in the virtual columns, or an unresolvable conflict, then zero rows are returned.

Determining Cycle Boundaries

Cycle boundaries are calculated based on the query start and end times, wwCycleCount, and wwResolution.

If you only specify wwCycleCount, evenly spaced cycles are returned based on the value of wwCycleCount.

If you only specify wwResolution, cycles are spaced wwResolution milliseconds apart starting at the query start time until query end time is reached. The last cycle will have whatever duration is required to end exactly at the query end time. If this last duration is shortened by this rule, it is known as a partial cycle. Because of this, the final cycle duration may not match wwResolution.

If both wwCycleCount and wwResolution are specified, no result rows will be returned. If you specify neither wwCycleCount nor wwResolution in the query, the query will return 100 rows.

Unless otherwise specified, a value is considered in a given full or partial cycle if its timestamp occurs at or after the cycle start (timestamp >= cycle start) and before the cycle end (timestamp < cycle end).

Mixing Tag Types in the Same Query

The History and Live tables use the sql_variant data type for the vValue column, allowing the return of various data types in a single column. In other words, these tables allow values for tags of different types to be retrieved with a simple query, without the need for a JOIN operation.

For example:

```
SELECT TagName, DateTime, vValue
   FROM History
      WHERE TagName IN ('SysTimeMin', 'SysPulse', 'SysString')
         AND DateTime >= '2001-12-20 0:00'
         AND DateTime <= '2001-12-20 0:05'
         AND wwRetrievalMode = 'delta'
```

The results are:

TagName	DateTime	vValue
SysTimeMin	2001-12-20 00:00:00.000	0
SysPulse	2001-12-20 00:00:00.000	0
SysString	2001-12-20 00:00:00.000	2001/12/20 08:00:00
SysTimeMin	2001-12-20 00:01:00.000	1
SysPulse	2001-12-20 00:01:00.000	1
SysTimeMin	2001-12-20 00:02:00.000	2
SysPulse	2001-12-20 00:02:00.000	0
SysTimeMin	2001-12-20 00:03:00.000	3
SysPulse	2001-12-20 00:03:00.000	1
SysTimeMin	2001-12-20 00:04:00.000	4
SysPulse	2001-12-20 00:04:00.000	0
SysTimeMin	2001-12-20 00:05:00.000	5
SysPulse	2001-12-20 00:05:00.000	1

Using a Criteria Condition on a Column of Variant Data

The AVEVA Historian OLE DB provider sends variant data to the SQL Server as a string. If the query contains a criteria condition on a column containing variant type data, the filtering is handled by SQL Server. An example of a criteria condition is:

```
WHERE ... vValue = 2
```

To perform the filtering, the SQL Server must determine the data type of the constant (in this example, 2), and attempt to convert the variant (string) to this destination type. The SQL Server assumes that a constant without a decimal is an integer, and attempts to convert the string to an integer type. This conversion will fail in SQL Server if the string actually represents a float (for example, 2.00123).

You should explicitly state the destination type by means of a CONVERT function. This is the only reliable way of filtering on the vValue column, which contains variant data.

For example:

```
SELECT DateTime, Quality, OPCQuality, QualityDetail, Value, vValue, TagName
   FROM History
      WHERE TagName IN ('ADxxxF36', 'SysTimeMin', 'SysPulse')
         AND DateTime >= '12-04-2001 04:00:00.000'
         AND DateTime <= '12-04-2001 04:03:00.000'
         AND wwRetrievalMode = 'Delta'
         AND convert(float, vValue) = 2
```

The following is another example:

```
SELECT DateTime, Quality, OPCQuality, QualityDetail, Value, vValue, TagName
   FROM History
      WHERE TagName IN ('VectorX', 'SysTimeMin', 'SysPulse')
         AND DateTime >= '20020313 04:00:07.000'
         AND DateTime <= '20020313 04:01:00.000'
         AND wwRetrievalMode = 'Delta'
         AND convert(float, vValue) > 1
         AND convert(float, vValue) < 2
```

Using DateTime Functions

Date functions perform an operation on a date and time input value and return either a string, numeric, or date and time value.

The following query returns the date/time stamp and value for the SysTimeSec tag for the last 10 minutes.

```
SELECT DateTime, TagName, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= dateadd(Minute, -10, GetDate())
         AND DateTime <= GetDate()
         AND wwRetrievalMode = 'Cyclic'
```

The results are:

DateTime	TagName	Value	Quality
2001-12-15 13:00:00.000	SysTimeSec	0.0	0
2001-12-15 13:00:06.060	SysTimeSec	6.0	0

2001-12-15 13:00:12.120	SysTimeSec	12.0	0
2001-12-15 13:00:18.180	SysTimeSec	18.0	0
2001-12-15 13:00:24.240	SysTimeSec	24.0	0
2001-12-15 13:00:30.300	SysTimeSec	30.0	0
2001-12-15 13:00:36.360	SysTimeSec	36.0	0
2001-12-15 13:00:42.420	SysTimeSec	42.0	0
...			

For any query, the SQL Server performs all date/time computations in local server time, reformulates the query with specific dates, and sends it on to the AVEVA Historian OLE DB provider. The AVEVA Historian OLE DB provider then applies the wwTimeZone parameter in determining the result set.

For example, the following query requests the last 30 minutes of data, expressed in Eastern Daylight Time (EDT). The server is located in the Pacific Daylight Time (PDT) zone.

```
SELECT DateTime, TagName, Value FROM History
    WHERE TagName IN ('SysTimeHour', 'SysTimeMin', 'SysTimeSec')
      AND DateTime > DateAdd(mi, -30, GetDate())
      AND wwTimeZone = 'eastern daylight time'
```

If it is currently 14:00:00 in the Pacific Daylight Time zone, then it is 17:00:00 in the Eastern Daylight Time zone. You would expect the query to return data from 16:30:00 to 17:00:00 EDT, representing the last 30 minutes in the Eastern Daylight Time zone.

However, the data that is returned is from 13:30:00 to 17:00:00 EDT. This is because the SQL Server computes the "`DateAdd(mi, -30, GetDate())`" part of the query assuming the local server time zone (in this example, PDT). It then passes the AVEVA Historian OLE DB provider a query similar to the following:

```
SELECT DateTime, TagName, Value FROM History
   WHERE TagName IN ('SysTimeHour', 'SysTimeMin', 'SysTimeSec')
      AND DateTime > 'YYYY-MM-DD 13:30:00.000'
      AND wwTimeZone = 'eastern daylight time'
```

Because the OLE DB provider is not provided an end date, it assumes the end date to be the current time in the specified time zone, which is 17:00:00 EDT.

Using the GROUP BY Clause

The GROUP BY clause works if the query uses the four-part naming convention or one of the associated views.

The following example will find the highest value of a specified set of tags over a time period.

```
SELECT TagName, Max(Value)
    FROM INSQL.Runtime.dbo.History
      WHERE TagName IN ('ReactTemp','ReactLevel','SysTimeSec')
        AND DateTime > '2001-12-20 0:00'
        AND DateTime < '2001-12-20 0:05'
        GROUP BY TagName
```

The results are:

SysTimeSec	59.0

Using the COUNT() Function

The COUNT(*) function works directly in a four-part query, but is not supported inside of the OPENQUERY function.

For example:

```
SELECT count(*)
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-12-20 0:00'
         AND DateTime <= '2001-12-20 0:05'
         AND wwRetrievalMode = 'delta'
         AND Value >= 30
```

The result is:

150	

If you use the OPENQUERY function, you cannot perform arithmetic functions on the COUNT(*) column. However, you can perform the count outside of the OPENQUERY, as follows:

```
SELECT count(*), count(*)/2  FROM OPENQUERY(INSQL, 'SELECT DateTime, vValue,
Quality, QualityDetail
   FROM History
      WHERE TagName IN ("SysTimeSec")
         AND DateTime >= "2002-04-16 03:00:00.000"
         AND DateTime <= "2002-04-16 06:00:00.000"
         AND wwRetrievalMode = "Delta"
   ')
```

The result is:

10801	5400
(1 row(s) affected)	

Using an Arithmetic Function

The following query adds the values of two tags from the WideHistory table.

```
SELECT * FROM OpenQuery(INSQL,'
   SELECT DateTime, ReactLevel, ProdLevel, "Sum" = ReactLevel+Prodlevel
      FROM WideHistory
         WHERE DateTime > "2001-02-28 18:56"
         AND DateTime < "2001-02-28 19:00"
         AND wwRetrievalMode = "Cyclic"
      ')
```

The results are:

DateTime	ReactLevel	Prodlevel	Sum
2001-02-28 18:56:00.000	1525.0	2343.0	3868.0
2001-02-28 18:56:00.000	1525.0	2343.0	3868.0
2001-02-28 18:56:00.000	1525.0	2343.0	3868.0
2001-02-28 18:56:00.000	1525.0	2343.0	3868.0
2001-02-28 18:56:00.000	2025.0	2343.0	4368.0
2001-02-28 18:56:00.000	2025.0	2343.0	4368.0
...			
(100 row(s) affected)			

If you use a math operator, such as plus (+), minus (-), multiply (*), or divide (/), you will need to add a blank space in front of and after the operator. For example, "Value - 2" instead of "Value-2".

Using an Aggregate Function

The following query returns the minimum, maximum, average, and sum of the tag 'ReactLevel' from the WideHistory table.

```
SELECT * FROM OpenQuery(INSQL,'
    SELECT "Minimum" = min(ReactLevel),
    "Maximum" = max(ReactLevel),
    "Average" = avg(ReactLevel),
    "Sum" = sum(ReactLevel)
        FROM WideHistory
            WHERE DateTime > "2001-02-28 18:55:00 "
            AND DateTime < "2001-02-28 19:00:00"
            AND wwRetrievalMode = "Cyclic"
    ')
```

The results are:

Minimum	Maximum	Average	Sum
-25.0	2025.0	1181.2	118120.0

(1 row(s) affected)

If you perform a SUM or AVG in delta retrieval mode against the Wide table, the aggregation will only be performed when the value has changed. The aggregation will not apply to all of the rows returned for each column.

For example, the following query has no aggregation applied:

```
SELECT * FROM OpenQuery(INSQL, 'SELECT DateTime, SysTimeHour, SysTimeMin,
SysTimeSec, SysDateDay
    FROM AnalogWideHistory
        WHERE DateTime >= "2001-08-15 13:20:57.345"
            AND DateTime < "2001-08-15 13:21:03.345"
            AND wwRetrievalMode = "Delta"
        ')
GO
```

The results are:

DateTime	SysTimeHour	SysTimeMin	SysTimeSec	SysDateDay
2001-08-15 13:20:57.343	13	20	57	15
2001-08-15 13:20:58.000	13	20	58	15
2001-08-15 13:20:59.000	13	20	59	15
2001-08-15 13:21:00.000	13	21	0	15
2001-08-15 13:21:01.000	13	21	1	15
2001-08-15 13:21:02.000	13	21	2	15
2001-08-15 13:21:03.000	13	21	3	15
(7 row(s) affected)				

Then, a SUM is applied to all of the returned column values:

```
SELECT * FROM OpenQuery(INSQL,'SELECT Sum(SysTimeHour), Sum(SysTimeMin),
Sum(SysTimeSec), Sum(SysDateDay)
   FROM WideHistory
      WHERE DateTime >= "2001-08-15 13:20:57.345"
         AND DateTime < "2001-08-15 13:21:03.345"
         AND wwRetrievalMode = "Delta"
      ')
GO
```

The results are:

SysTimeHour	SysTimeMin	SysTimeSec	SysDateDay
13	41	180	15

Thus, for delta retrieval mode, a SUM or AVG is applied only if the value has changed from the previous row.

If you perform an AVG in delta retrieval mode, AVG will be computed as:

SUM of delta values/number of delta values

For example, an AVG is applied to all of the returned column values:

```
SELECT * FROM OpenQuery(INSQL,'SELECT Avg(SysTimeHour), Avg(SysTimeMin),
Avg(SysTimeSec), Avg(SysDateDay)
   FROM WideHistory
      WHERE DateTime >= "2001-08-15 13:20:57.345"
         AND DateTime < "2001-08-15 13:21:03.345"
         AND wwRetrievalMode = "Delta"
      ')
GO
```

The results are:

SysTimeMin	SysTimeSec
20.5	25.714285714285715

Making and Querying Annotations

The following query creates an annotation for the specified tag. The annotation is made in response to a pump turning off. Then, the annotations for a particular tag are returned.

```
DECLARE @@UserKey INT
SELECT @@UserKey = UserKey
   FROM UserDetail
      WHERE UserName = 'wwAdmin'
INSERT INTO Annotation (TagName, UserKey, DateTime, Content)
      VALUES ('ReactLevel', @@UserKey, GetDate(), 'The Pump is off')
SELECT DateTime, TagName, Content
   FROM Annotation
      WHERE Annotation.TagName = 'ReactLevel'
         AND DateTime > '27 Feb 01'
         AND DateTime <= GetDate()
```

The results are:

DateTime	TagName	Content
2001-02-28 19:18:00.000	ReactLevel	The Pump is off
(1 row(s) affected)		

Using Comparison Operators with Delta Retrieval

The system behaves differently when doing typical delta-based queries where a start date and end date are specified using the comparison operators >=, >, <= and <. The comparison operators can be used on the History and WideHistory tables. The comparison operators also apply regardless of how the query is executed (for example, four-part naming, OLE DB provider views, and so on).

Delta queries that use the comparison operators return all the valid changes to a set of tags over the specified time span. Using deadbands and other filters may modify the set of valid changes.

Specifying the Start Date with ">="

If the start date is specified using >= (greater than or equal to), then a row is always returned for the specified start date. If the start date/time coincides exactly with a valid value change, then the Quality is normal (0). Otherwise, the value at the start date is returned, and the Quality value is 133 (because the length of time that the tag's value was at *X* is unknown).

Query 1

For this query, the start date will not correspond to a data change:

```
SELECT DateTime, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeMin'
         AND wwRetrievalMode = 'Delta'
         AND DateTime >= '2001-01-13 12:00:30'
         AND DateTime < '2001-01-13 12:10:00'
```

The start time (12:00:30) does not correspond with an actual change in value, and is therefore marked with the initial quality of 133:

DateTime	Value	Quality
2001-01-13 12:00:30.000	0	133
2001-01-13 12:01:00.000	1	0
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0

```
(10 row(s) affected)
```

Query 2

For this query, the start date will correspond to a data change:

```
SELECT DateTime, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeMin'
         AND wwRetrievalMode = 'Delta'
         AND DateTime >= '2001-01-13 12:01:00'
         AND DateTime < '2001-01-13 12:10:00'
```

The start time (12:01:00) does correspond exactly with an actual change in value, and is therefore marked with the normal quality of 0.

DateTime	Value	Quality
2001-01-13 12:01:00.000	1	0
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0

```
(9 row(s) affected)
```

Query 3

For this query, the start date will return at least one row, even though the query captures no data changes:

```
SELECT DateTime, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeMin'
         AND wwRetrievalMode = 'Delta'
         AND DateTime >= '2001-01-13 12:00:30'
         AND DateTime < '2001-01-13 12:01:00'
```

The query does not capture an actual change in value, and is therefore marked with the initial value quality of 133 for the start time of the query:

DateTime	Value	Quality
2001-01-13 12:00:30.000	0	133

```
(1 row(s) affected)
```

Specifying the Start Date with ">"

If the start date is specified using > (greater than), then the first row returned is the first valid change after (but not including) the start date. No initial value row is returned. A query that uses > to specify its start date may return zero rows.

Query 1

For this query, the first row that will be returned will be the first valid change after (but not including) the start time (12:00:30):

```
SELECT DateTime, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeMin'
         AND wwRetrievalMode = 'Delta'
         AND DateTime > '2001-01-13 12:00:30'
         AND DateTime < '2001-01-13 12:10:00'
```

The first row returned is the first valid change after (but not including) the start time (12:00:30):

DateTime	Value	Quality
2001-01-13 12:01:00.000	1	0
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0

```
(9 row(s) affected)
```

Query 2

For this query, the start date will correspond to a data change, but it will be excluded from the result set because the operator used is greater than, not greater than or equal to.

```
SELECT DateTime, Value, Quality
```

```
FROM History
    WHERE TagName = 'SysTimeMin'
        AND wwRetrievalMode = 'Delta'
        AND DateTime > '2001-01-13 12:01:00'
        AND DateTime < '2001-01-13 12:10:00'
```

The start time (12:01:00) corresponds exactly with an actual change in value, but it is excluded from the result set because the operator used is greater than, not greater than or equal to.

DateTime	Value	Quality
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0

```
(8 row(s) affected)
```

Query 3

This query will return no rows, because no data changes are captured:

```
SELECT DateTime, Value, Quality
    FROM History
        WHERE TagName = 'SysTimeMin'
            AND wwRetrievalMode = 'Delta'
            AND DateTime > '2001-01-13 12:00:30'
            AND DateTime < '2001-01-13 12:01:00'
```

The query does not capture an actual change in value; therefore, no rows are returned.

DateTime	Value	Quality

```
(0 row(s) affected)
```

Specifying the End Date with "<="

If the end date is specified using <= (less than or equal to) then the last row returned is the last valid change up to, and including, the end date. If the end date uses "<=" then the last change returned may have a date/time exactly at the end date. If there is a value exactly at the end date, it will be returned.

This query uses the remote table view.

```
SELECT DateTime, Value, Quality
    FROM History
        WHERE TagName = 'SysTimeMin'
            AND wwRetrievalMode = 'Delta'
            AND DateTime > '2001-01-13 12:00:30'
            AND DateTime <= '2001-01-13 12:10:00'
```

Note that there is a valid change at exactly the end time of the query (12:10:00):

DateTime	Value	Quality
2001-01-13 12:01:00.000	1	0
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0
2001-01-13 12:10:00.000	10	0

```
(10 row(s) affected)
```

Specifying the End Date with "<"

If the end date is specified using < (less than), then the last row returned is the last valid change up to (but not including) the end date. If the end date uses "<" then the last event returned will have a date/time less than the end date. If there is an event exactly at the end date, it will not be returned.

This query uses the remote table view.

```
SELECT DateTime, Value, Quality
   FROM History
      WHERE TagName = 'SysTimeMin'
         AND wwRetrievalMode = 'Delta'
         AND DateTime > '2001-01-13 12:00:30'
         AND DateTime < '2001-01-13 12:10:00'
```

Note that there is a valid change at exactly the end time of the query (12:10:00), but it is excluded from the result set.

DateTime	Value	Quality
2001-01-13 12:01:00.000	1	0
2001-01-13 12:02:00.000	2	0
2001-01-13 12:03:00.000	3	0
2001-01-13 12:04:00.000	4	0
2001-01-13 12:05:00.000	5	0
2001-01-13 12:06:00.000	6	0
2001-01-13 12:07:00.000	7	0
2001-01-13 12:08:00.000	8	0
2001-01-13 12:09:00.000	9	0

```
(9 row(s) affected)
```

Using Comparison Operators with Cyclic Retrieval and Cycle Count

Cyclic queries with the wwCycleCount time domain extension return a set of evenly spaced rows over the specified time span. The result set will always return the number of rows specified by the cycle count extension for each tag in the query. The resolution for these rows is calculated by dividing the time span by the cycle count.

Specifying Cycle Count with Two Equity Operators

If the time range is specified using >= and <=, then the first row falls exactly on the start time, and the last row falls exactly on the end time. In this case, the resolution used is (end date – start date) / (cyclecount – 1).

This query uses a cycle count of 60, resulting in a 1 second resolution for the data. The query uses the remote table view.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2001-01-13 12:00:00'
         AND DateTime <= '2001-01-13 12:00:59'
         AND wwCycleCount = 60
         AND wwRetrievalMode = 'Cyclic'
```

The results are:

DateTime	Value
2001-01-13 12:00:00.000	0
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59

```
(60 row(s) affected)
```

Specifying Cycle Count with One Equity Operator

If one end of the time range is excluded (by using > instead of >= or < instead of <=). then a gap of "resolution" is left at the beginning (or end) of the result set.

The resolution is calculated as (end date – start date) / (cyclecount).

The row that equates to the time which is designated using the < (or >) operator is not returned.

These queries use the remote table view.

Query 1

This query uses a cycle count of 60, resulting in a 1 second resolution for the data. The starting time is set to >=.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwCycleCount = 60
         AND DateTime >= '2001-01-13 12:00:00'
         AND DateTime < '2001-01-13 12:01:00'
```

The results are:

DateTime	Value
2001-01-13 12:00:00.000	0
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59

```
(60 row(s) affected)
```

Query 2

This query also uses a cycle count of 60, resulting in a 1 second resolution for the data. The ending time is set to <=.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwCycleCount = 60
         AND DateTime > '2001-01-13 12:00:00'
         AND DateTime <= '2001-01-13 12:01:00'
```

The results are:

DateTime	Value
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3

2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59
2001-01-13 12:01:00.000	0

(60 row(s) affected)

Specifying Cycle Count with No Equity Operators

If both ends of the time range are excluded (by using > and <) then a gap of "resolution" is left at the beginning and end of the result set.

The resolution is calculated as (end date – start date) / (cyclecount + 1).

The row(s) that equate to the start and end times are not returned.

This query uses the remote table view.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwCycleCount = 60
         AND DateTime > '2001-01-13 12:00:00'
         AND DateTime < '2001-01-13 12:01:01'
```

The results are:

DateTime	Value
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59
2001-01-13 12:01:00.000	0

(60 row(s) affected)

Using Comparison Operators with Cyclic Retrieval and Resolution

Cyclic queries that use comparison operators and the resolution time domain extension return a set of evenly spaced rows over the specified time span. The resolution for these rows is specified in the query.

Using Two Equality Operators for Comparison with Cyclic Retrieval and Resolution

If the time range is specified using >= and <=, then the first row falls exactly on the start time. The last row will fall exactly on the end time, if the resolution divides exactly into the specified time duration. If the resolution does not divide exactly into the specified time duration, then the last row returned will be the last row satisfying (start date + N*resolution) which has a timestamp less than the end date.

In short:

- <= endtime MAY return a last row containing the exact endtime (but it is not guaranteed to do so)
- < endtime is guaranteed NOT to return a last row containing the exact endtime

This query sets the resolution to 1 second.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwResolution = 1000
         AND DateTime >= '2001-01-13 12:00:00'
         AND DateTime <= '2001-01-13 12:01:00'
```

The results are:

DateTime	Value
2001-01-13 12:00:00.000	0
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59
2001-01-13 12:01:00.000	0

(61 row(s) affected)

Using One Equality Operator for Comparison with Cyclic Retrieval and Resolution

If the start time is excluded (by using > instead of >=), then a gap of "resolution" is left at the beginning of the result set. In this case, the first row returned will have the timestamp of the (start date + resolution). If the end date uses "<" then the last row returned will be the last row defined by (start date + N*resolution) which has a timestamp less than the end date.

The row that equates to the time that is designated using the < (or >) operator is not returned.

Query 1

This query uses a resolution of 1000, resulting in a 1 second resolution for the data.. The starting time is set to >=.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwResolution = 1000
         AND DateTime >= '2001-01-13 12:00:00'
         AND DateTime < '2001-01-13 12:01:00'
```

The results are:

DateTime	Value
2001-01-13 12:00:00.000	0
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:55.000	55
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59

```
(60 row(s) affected)
```

Query 2

This query also uses a row resolution of 1000, resulting in a 1 second resolution for the data. The starting time is set to <=.

```
SELECT DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND wwResolution = 1000
         AND DateTime > '2001-01-13 12:00:00'
         AND DateTime <= '2001-01-13 12:01:00'
```

The results are:

DateTime	Value
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59
2001-01-13 12:01:00.000	0

(60 row(s) affected)

Using No Equality Operators for Comparison with Cyclic Retrieval and Resolution

If both ends of the time range are excluded (by using > and <), then a gap of resolution is left at the beginning and end of the result set.

The row(s) that equate to the start and end times are not returned.

This query uses a resolution of 1000, resulting in a 1 second resolution for the data.

```
SELECT DateTime, Value
   FROM v_AnalogHistory
      WHERE TagName = 'SysTimeSec'
         AND wwResolution = 1000
         AND DateTime > '2001-01-13 12:00:00'
         AND DateTime < '2001-01-13 12:01:01'
```

The results are:

DateTime	Value
2001-01-13 12:00:01.000	1
2001-01-13 12:00:02.000	2
2001-01-13 12:00:03.000	3
2001-01-13 12:00:04.000	4
...	
2001-01-13 12:00:56.000	56
2001-01-13 12:00:57.000	57
2001-01-13 12:00:58.000	58
2001-01-13 12:00:59.000	59

DateTime	Value
2001-01-13 12:01:00.000	0

(60 row(s) affected)

Returning Time Between Value Changes

You can return the amount of time before a tag's value changed to a subsequent value. This time is returned using the wwResolution column.

This functionality works with the cyclic, delta, and full retrieval modes. The delta and full mode behavior of wwResolution does not apply to the AnalogSummaryHistory and StateSummaryHistory tables.

If the time change value is greater than 2,147,000,000 milliseconds (~25 days), then the value of wwResolution column is -1.

Example 1: Cyclic Retrieval

For this example, the following data is stored in the Historian:

DateTime	Value
2012-01-01 07:59:53	34.42384
2012-01-01 08:00:13	15.02637
2012-01-01 08:00:33	20.29732
2012-01-01 08:00:53	37.40273
2012-01-01 08:01:13	24.31662

For cyclic retrieval, cycles start at the start DateTime and occur at intervals specified by wwResolution in the query. If you query for the data 2012-01-01 08:00:00 to 2012-01-01 08:01:00 with four cycles, the wwResolution column in the results show the time in milliseconds until the next point.

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	20000
2012-01-01 08:00:20	15.02637	20000
2012-01-01 08:00:40	20.29732	20000
2012-01-01 08:01:00	37.40273	20000

Example 2: Delta and Full Retrieval

In delta and full mode, the wwResolution column is used in the query results to show the time in milliseconds until the next point. When the first point in the result occurs before the query start time, the wwResolution column shows the time from the start of the query to the next point. If there are no more points after the first point, the wwResolution column is NULL.

For example, the following data is stored:

DateTime	Value
2012-01-01 07:59:53	34.42384

If you query for the data between 2012-01-01 08:00:00 to 2012-01-01 08:00:10, the results are:

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	NULL

However, if the data stored is:

DateTime	Value
2012-01-01 07:59:53	34.42384
2012-01-01 08:00:13	15.02637

Then the results are:

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	10000

When the last point in the result occurs before the query end time, the wwResolution column shows the time until the end of the query when there is a next available point. If there are no more points, then the wwResolution column shows NULL.

For example, the following is stored:

DateTime	Value
2012-01-01 07:59:53	34.42384
2012-01-01 08:00:13	15.02637
2012-01-01 08:00:33	20.29732
2012-01-01 08:00:53	37.40273
2012-01-01 08:01:13	24.31662

If you query for data from 2012-01-01 08:00:00 to 2012-01-01 08:05:00, the results are:

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	13000
2012-01-01 08:00:13	15.02637	20000
2012-01-01 08:00:33	20.29732	20000

DateTime	Value	wwResolution
2012-01-01 08:00:53	37.40273	20000
2012-01-01 08:01:13	24.31662	NULL

If you query for data from 2012-01-01 08:00:00 to 2012-01-01 08:01:00, the results are:

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	13000
2012-01-01 08:00:13	15.02637	20000
2012-01-01 08:00:33	20.29732	20000
2012-01-01 08:00:53	37.40273	7000

If the last point happens to be end of the query, then the wwResolution value is zero, even when there are no more points after the last point. For example, if you query for data from 2012-01-01 08:00:00 to 2012-01-01 08:01:13, the results are:

DateTime	Value	wwResolution
2012-01-01 08:00:00	34.42384	13000
2012-01-01 08:00:13	15.02637	20000
2012-01-01 08:00:33	20.29732	20000
2012-01-01 08:00:53	37.40273	20000
2012-01-01 08:01:13	24.31662	0

Example 3: Querying the WideHistory Table

If you execute a query on the WideHistory table for analog tags, wwResolution shows the time between the first value change for ANY of the tags.

```
SELECT * FROM OpenQuery(INSQL,'
   SELECT DateTime, SysTimeSec, SysTimeMin, wwResolution
      FROM WideHistory
         WHERE DateTime >= "20120119 12:44:00.000"
         AND DateTime <= "20120119 12:45:00.000"
         AND wwRetrievalMode = "Delta"
   ')
```

The results are:

DateTime	SysTimeSec	SysTimeMin	wwResolution
2012-01-19 12:44:00.0000000	0	44	1000
2012-01-19 12:44:01.0000000	1	44	1000
2012-01-19 12:44:02.0000000	2	44	1000

DateTime	SysTimeSec	SysTimeMin	wwResolution
2012-01-19 12:44:03.0000000	3	44	1000
2012-01-19 12:44:04.0000000	4	44	1000
2012-01-19 12:44:05.0000000	5	44	1000
...			

The wwResolution column shows 1000 milliseconds because the smallest time change is for the SysTimeSec tag, which is changing every second.

If you run the same query using the SysTimeHour tag instead of the SysTimeSec tag, the results are:

DateTime	SysTimeHour	SysTimeMin	wwResolution
2012-01-19 12:44:00.0000000	12	44	60000
2012-01-19 12:45:00.0000000	12	45	0

The wwResolution column shows 60000 milliseconds because the smallest time change is for the SysTimeMin tag, which is changing every minute (every 60 seconds). Because the query ended at the time of the last value, a 0 is shown for wwResolution for the ending value.

Example 4: Querying the History Table with the wwValueSelector Parameter

You can query the History table with the wwValueSelector parameter.

```
SELECT DateTime, TagName, Value, wwResolution
   FROM History
      WHERE Tagname like 'MyTag'
      AND DateTime >= '2012-01-19 10:00:00'
      AND DateTime <= '2012-01-19 11:00:00'
      AND wwValueSelector = 'STDDEV'
```

The results are:

DateTime	TagName	Value	wwResolution
2012-01-19 10:00:00.0000000	MyTag	977.157928752564	60000
2012-01-19 10:01:00.0000000	MyTag	16.5619987924163	60000
2012-01-19 10:02:00.0000000	MyTag	16.5619987924163	60000
2012-01-19 10:03:00.0000000	MyTag	16.5619987924165	60000
2012-01-19 10:04:00.0000000	MyTag	16.5619987924163	180000

DateTime	TagName	Value	wwResolution
2012-01-19 10:07:00.0000000	MyTag	16.5619987924171	60000
2012-01-19 10:08:00.0000000	MyTag	16.5619987924163	60000
2012-01-19 10:09:00.0000000	MyTag	16.5619987924179	60000
2012-01-19 10:10:00.0000000	MyTag	16.5619987924163	180000
2012-01-19 10:13:00.0000000	MyTag	16.5619987924195	60000
2012-01-19 10:14:00.0000000	MyTag	16.5619987924163	240000
2012-01-19 10:18:00.0000000	MyTag	16.5619987924226	180000
2012-01-19 10:21:00.0000000	MyTag	16.5619987924163	60000
2012-01-19 10:22:00.0000000	MyTag	16.5619987924226	60000
2012-01-19 10:23:00.0000000	MyTag	16.5619987924038	60000
...			

Example 5: Calculating Total Time Between Value Changes

You can calculate the total time the multiple discrete tags are in a certain state. For example, you want to know the total time that two pumps were on during a 24-hour period.

The following query returns a dataset that shows the time when both discrete tags had a value of 0:

```
SELECT * FROM OPENQUERY(INSQL,'
   SELECT DateTime, Pump1, Pump2, wwResolution
      FROM WideHistory
         WHERE DateTime >= "2012-03-08 16:00"
         AND DateTime < "2012-03-08 17:00"
         AND wwRetrievalMode="DELTA"
   ')
WHERE Pump1+Pump2=0
```

The results are:

DateTime	Pump1	Pump2	wwResolution
2012-03-08 16:00:00.0000000	0	0	67
2012-03-08 16:00:01.5980000	0	0	2521
2012-03-08 16:00:04.4470000	0	0	18500
2012-03-08 16:00:23.6000000	0	0	13995

DateTime	Pump1	Pump2	wwResolution
2012-03-08 16:00:37.9140000	0	0	2625
...			

The following query shows how to return the total time when both tags had a value of 0:

```
SELECT SUM(wwResolution) FROM OPENQUERY(INSQL,'
   SELECT DateTime, Total=Pump1+Pump2, wwResolution
      FROM WideHistory
         WHERE DateTime >= "2012-03-08 16:00"
         AND DateTime < "2012-03-08 17:00"
         AND wwRetrievalMode="DELTA"
   ')
WHERE Total=0
```

The results are:

(No column name)
2551289

If you changed the ending WHERE clause to Total>0, the returned time would be for when more than one discrete tag was true.

SELECT INTO from a History Table

The following query inserts the specified data from the WideHistory table into another table called MyTable. Then, the data in the MyTable table is queried. This query uses the OPENQUERY function.

```
DROP TABLE MyTable
SELECT DateTime,
      "Sec" = datepart(ss, DateTime),
      "mS"  = datepart(ms, DateTime),
      ReactTemp, ReactLevel
   INTO MyTable
      FROM OpenQuery(INSQL, 'SELECT DateTime, ReactTemp, ReactLevel FROM
      WideHistory
         WHERE wwResolution = 5000
         AND DateTime >= "2001-03-13 1:58pm"
         AND DateTime <= "2001-03-13 2:00pm" ')
SELECT * FROM MyTable
```

The results are:

DateTime	Sec	mS	ReactTemp	ReactLevel
2001-03-13 13:58:00.000	0	0	190.9	2025.0
2001-03-13 13:58:00.000	5	0	190.9	2025.0
2001-03-13 13:58:00.000	10	0	168.3	1215.0
2001-03-13 13:58:00.000	15	0	168.3	1215.0
2001-03-13 13:58:00.000	20	0	133.8	315.0
2001-03-13 13:58:00.000	25	0	133.8	315.0

2001-03-13 13:58:00.000	30	0	101.6	0.0
2001-03-13 13:58:00.000	35	0	101.6	0.0
2001-03-13 13:58:00.000	40	0	32.4	750.0
2001-03-13 13:58:00.000	45	0	32.4	750.0
2001-03-13 13:58:00.000	50	0	20.9	1700.0
2001-03-13 13:58:00.000	55	0	20.9	1700.0
2001-03-13 13:59:00.000	0	0	85.9	2000.0
2001-03-13 13:59:00.000	5	0	85.9	2000.0
2001-03-13 13:59:00.000	10	0	185.9	2000.0
2001-03-13 13:59:00.000	15	0	185.9	2000.0
2001-03-13 13:59:00.000	20	0	168.3	1235.0
2001-03-13 13:59:00.000	25	0	168.3	1235.0
2001-03-13 13:59:00.000	30	0	136.1	335.0
2001-03-13 13:59:00.000	35	0	136.1	335.0
2001-03-13 13:59:00.000	40	0	103.9	-25.0
2001-03-13 13:59:00.000	45	0	103.9	-25.0
2001-03-13 13:59:00.000	50	0	34.7	625.0
2001-03-13 13:59:00.000	55	0	34.7	625.0
2001-03-13 14:00:00.000	0	0	20.9	1575.0

(25 row(s) affected)

Moving Data from a SQL Server Table to an Extension Table

The following queries show how to insert manual data into a normal SQL Server table and then move it into the History extension table.

First, insert the data into the SQL Server table. The following query inserts two minutes of existing data for the SysTimeSec tag into the ManualAnalogHistory table:

```
INSERT INTO ManualAnalogHistory (DateTime, TagName, Value, Quality,
QualityDetail, wwTagKey)
   SELECT DateTime, TagName, Value, Quality, QualityDetail, wwTagKey
      FROM History WHERE TagName = 'SysTimeSec'
         AND DateTime >= '20050329 12:00:00'
         AND DateTime <= '20050329 12:02:00'
```

Then, create a manual tag using the System Management Console. For a manual tag, "MDAS/Manual Acquisition" is specified as the acquisition type. Be sure to commit the changes to the system. In this example, a manual analog tag named MDAS1 was created.

Finally, insert the data from the ManualAnalogHistory table into History:

```
INSERT INTO History (TagName, DateTime, Value, QualityDetail)
   SELECT 'MDAS1', DateTime, Value, QualityDetail FROM ManualAnalogHistory
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '20050329 12:00:00'
```

```
AND DateTime <= '20050329 12:02:00'
```

Using Server-Side Cursors

Cursors are a very powerful feature of SQL Server. They permit controlled movement through a record set that results from a query.

For in-depth information on cursors, see your Microsoft SQL Server documentation.

The AVEVA Historian OLE DB Provider provides server-side cursors. Cursors can be used to do joins that are not possible in any other way. They can be used to join date/times from any source with date/times in the history tables.

The following query provides an example of using a server-side cursor. This query:

- Fetches all of the events in the EventHistory table.
- Shows a "snapshot" of three tags at the time of each event.
- Shows the event tag and its associated key value.

This query could easily be encapsulated into a stored procedure. The query uses the four-part naming convention.

```
SET QUOTED_IDENTIFIER OFF
DECLARE @DateValue DateTime
DECLARE @EventTag nvarchar(256)
DECLARE @EventKey int
DECLARE @Qry1 nvarchar(500)
DECLARE @Qry2 nvarchar(500)
DECLARE @Qry3 nvarchar(500)
SELECT @Qry1 = N'SELECT EventTag = @EventTag, EventKey = @EventKey, DateTime,
TagName, Value, Quality
   FROM History
      WHERE TagName IN (N''SysTimeSec'', N''SysTimeMin'', N''SysTimeHour'')
         AND DateTime = '''
SELECT @Qry2 = N''''
SELECT @Qry3 = N''
DECLARE Hist_Cursor CURSOR FOR
SELECT DateTime, TagName, EventLogKey
      FROM Runtime.dbo.EventHistory
OPEN Hist_Cursor
FETCH NEXT FROM Hist_Cursor INTO @DateValue, @EventTag, @EventKey
WHILE @@FETCH_STATUS = 0
BEGIN
   SELECT @Qry3 = @Qry1 + convert(nvarchar, @DateValue, 121) + @Qry2
   --PRINT @Qry3
   EXEC sp_executesql @Qry3, N'@EventTag nvarchar(256),
   @EventKey int', @EventTag, @EventKey
   FETCH NEXT FROM Hist_Cursor INTO @DateValue, @EventTag, @EventKey
END
CLOSE Hist_Cursor
DEALLOCATE Hist_Cursor
```

The results are:

EventTag	Event Key	DateTime	TagName	Value	Quality
SysStatusEvent	3	2001-01-12 13:00:27.000	SysTimeSec	27.0	0

SysStatusEvent	3	2001-01-12 13:00:27.000	SysTimeMin	0.0	0
SysStatusEvent	3	2001-01-12 13:00:27.000	SysTimeHour	13.0	0

(3 row(s) affected)

EventTag	Event Key	DateTime	TagName	Value	Quality
SysStatusEvent	4	2001-01-12 14:00:28.000	SysTimeSec	28.0	0
SysStatusEvent	4	2001-01-12 14:00:28.000	SysTimeMin	0.0	0
SysStatusEvent	4	2001-01-12 14:00:28.000	SysTimeHour	14.0	0

(3 row(s) affected)

Using Stored Procedures in OLE DB Queries

Any normal SQL Server stored procedure can make use of the tables exposed by the AVEVA Historian OLE DB Provider. Stored procedures can use any valid Transact-SQL syntax to access AVEVA Historian historical data.

In other words, stored procedures can make use of four-part-queries, OPENQUERY and OPENROWSET functions, cursors, parameterized queries and views. Stored procedures can be used to encapsulate complex joins and other operations for easy re-use by applications and end users.

Getting Data from the OPCQualityMap Table

In general, an OPC quality has 16 significant bits. The lower 8 bits contain the quality as described in the table, while the upper 8 bits hold server-specific information. To ensure correct results, it is important to consider only the lower 8 bits in a query or join involving the OPCQualityMap table.

For example:

```
SELECT h.DateTime, h.TagName, h.Value, o.Description FROM History h
     INNER JOIN OPCQualityMap o
     ON (h.OPCQuality & 255) = o.OPCQuality
        WHERE TagName in (…)
        AND …
```

Using Variables with the Wide Table

You cannot use variables in an OPENQUERY statement. Therefore, if you want to use variables in a query on the wide table, you must first build up the OPENQUERY statement "on the fly" as a string, and then execute it.

```
DECLARE @sql nvarchar(1000)
DECLARE @DateStart datetime

DECLARE @DateEnd datetime

SET @DateStart = '2001-8-29 11:00:00'
```

```
SET @DateEnd = '2001-8-29 11:11:00'

SET @sql = N'select *

FROM OPENQUERY(INSQL, ''SELECT DateTime, ReactLevel, ReactTemp, ProdLevel,
BatchNumber, ConcPump, Mixer, TransferValve, TransferPump, WaterValve,
ConcValve, OutputValve, SteamValve
   FROM WideHistory
      WHERE DateTime >= "' + CONVERT(varchar(26), @DateStart, 113) + '"
            and DateTime <=  "' + CONVERT(varchar(26), @DateEnd, 113) + '"
            AND wwResolution = 1000
            AND wwRetrievalMode = "cyclic"'') '

EXEC sp_executesql @sql
```

Retrieval Across a Data Gap in Classically Stored Data

For blocks created by the Classic Storage subsystem, if the data to be retrieved spans more than one history block, and the start time of the later block is equal (within one tick) to the end time of the first block, you will not notice any difference than when querying within a single block.

However, if the system was stopped between history blocks, there will be a gap in the data, as shown in the following diagram:

Upon retrieval, additional data points (labeled A and B) will be added to mark the end of the first block's data and the beginning of the second block's data. Point C is a stored point generated by the Storage subsystem. (Upon a restart, the first value from each IDAS will be offset from the start time by 2 seconds and have a quality detail of 252.)

The following paragraphs explain this in more detail.

For delta retrieval, the data values in the first block are returned as stored. After the end of the block is reached and all of the points have been retrieved, an additional data point (A) will be inserted by retrieval to mark the end of the data. The value for point A will be

Point A attribute	Value(Hex)	Value(Dec)
Value	0	0
Quality	100	256

Quality Detail	0	0

If there is no value stored at the beginning of the next block, an initial data point (B) will be inserted by retrieval and will have the snapshot initial value as stored. The quality and quality detail values are as follows:

Point B attribute	Value(Hex)	Value(Dec)
Value	Snapshot	Snapshot
Quality	0	0
Quality Detail	96	150

In the case of cyclic retrieval, a point is required for each specified time. If the time coincides with the data gap, a NULL point for that time will be generated. The inserted points will have the values defined in the following table.

Cyclic NULL point	Value(Hex)	Value(Dec)
Value	0	0
Quality	100	256
Quality Detail	0	0

If you are using time or value deadbands for delta retrieval across a data gap, the behavior is as follows:

- For a value deadband, all NULLs will be returned and all values immediately after a NULL will be returned. That is, the deadband is not applied to values separated by a NULL.
- For a time deadband, null values are treated like any other value. Time deadbands are not affected by NULLs.

Returned Values for Non-Valid Start Times

One example of a non-valid query start time is a start time that is earlier than the start time of the first history block. For delta retrieval, the first row returned will be NULL. The timestamp will be that of the query start time. The next row returned will be timestamped at the start of the history block and have the following attributes:

Point attribute	Value(Hex)	Value(Dec)
Value	Snapshot	Snapshot
Quality	0	0
Quality Detail	96	150

For cyclic retrieval, NULL will be returned for data values that occur before the start of the history block.

Another non-valid start time is a start time that is later than the current time of the AVEVA Historian computer. For delta retrieval, a single NULL value will be returned. For cyclic retrieval, a NULL will be returned for each data value requested.

Querying Aggregate Data in Different Ways

There are four different ways you can retrieve summary data, such as an average, using the Historian.

- Using the SQL Server average function. This is appropriate for discrete samples. For example, a check weigher, where you are measuring individual units against a target weight.
- Using the average retrieval mode. This is appropriate for most situations where you want to find an average, as it is weighted according to time. For example, if you want to find the average for a flow rate or a temperature.
- Setting up summary replication and then querying the AnalogSummaryHistory table. Replication uses the average retrieval mode to do the calculations.
- Setting up a summary event and then querying the SummaryData table. The Event subsystem uses the SQL Server average function.

The following examples show how you can get the same data using these different methods. All examples use the SysTimeSec system tag, which has a range of 0 to 59.

Query 1

The following query uses the SQL Server average function to return the average value of the SysTimeSec tag over the span of one minute.

```
SELECT AVG(Value) as 'SysTimeSec AVG'
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime > '2009-11-15 6:30:00'
         AND DateTime < '2009-11-15 6:31:00'
         AND wwRetrievalMode = 'Full'
```

The results are:

SysTimeSec AVG
29.5

Query 2

The following query uses the historian time-weighted average retrieval mode to return the average for the same time period. Because the cycle count is set to 2, a first row is returned for the "phantom"cycle leading up to the query start time. The StartDateTime column shows the time stamp at the start of the data sampling, which is the start time of the phantom cycle. The second row returned reflects is the actual data that you expect. The time stamp for the data value is 2009-11-15 06:31:00 because the default time stamping rule is set so that the ending time stamp for the cycle is returned. For more information about the phantom cycle, see *About Phantom Cycles* on page 95.

```
SELECT StartDateTime, DateTime, TagName, Value
   FROM History
      WHERE TagName = 'SysTimeSec'
         AND DateTime >= '2009-11-15 6:30:00'
         AND DateTime <= '2009-11-15 6:31:00'
         AND wwRetrievalMode = 'Average'
         AND wwCycleCount = 2
         AND wwTimeStampRule = 'end'
```

The results are:

StartDateTime	DateTime	TagName	Value
2009-11-15 06:29:00	2009-11-15 06:30:00	SysTimeSec	29.5
2009-11-15 06:30:00	2009-11-15 06:31:00	SysTimeSec	29.5

Query 3

For the following query, local replication has been set up so that the average of the SysTimeSec tag is calculated every minute and stored to the SysTimeSec.1M analog summary tag. The query returns the value of the SysTimeSec.1M tag for the time period specified.

```
SELECT TagName, StartDateTime, EndDateTime, Average as AVG
   FROM AnalogSummaryHistory
      WHERE TagName = 'SysTimeSec.1M'
         AND StartDateTime >= '2009-11-15 6:30:00'
         AND EndDateTime <= '2009-11-15 6:31:00'
```

The results are:

TagName	StartDateTime	EndDateTime	AVG
SysTimeSec.1M	2009-11-15 06:30:00	2009-11-15 06:31:00	29.5

Query 4

The following query, the History table is used instead of the AnalogSummaryHistory table. Because the cycle count is set to 2, this query returns a row for the phantom cycle. The time stamp for the data value is 2009-11-15 06:31:00 because the default time stamping rule is set so that the ending time stamp for the cycle is returned.

```
SELECT TagName, DateTime, Value
   FROM History
      WHERE TagName = 'SysTimeSec.1M'
         AND DateTime >= '2009-11-15 6:30:00'
         AND DateTime <= '2009-11-15 6:31:00'
         AND wwRetrievalMode = 'avg'
         AND wwCycleCount = 2
```

The results are:

TagName	DateTime	Value
SysTimeSec.1M	2009-11-15 06:30:00	29.5
SysTimeSec.1M	2009-11-15 06:31:00	29.5

Query 5

The following query returns five minutes of summary data for an event tag that has been configured to store the average value of the SysTimeSec tag every minute.

```
SELECT TagName, CalcType, SummaryDate, Value
   FROM v_SummaryData
      WHERE TagName = 'SysTimeSec'
         AND SummaryDate >= '2009-11-15 18:30:00'
         AND SummaryDate <= '2009-11-15 18:31:00'
```

The results are:

TagName	CalcType	SummaryDate	Value
SysTimeSec	AVG	2009-11-15 18:30:00.000	29.5
SysTimeSec	AVG	2009-11-15 18:31:00.000	29.5

Bitwise Retrieval for Process Data

It is common to pack multiple digital states into the same PLC register as an integer rather than as individual bits. You can still map the individual bits to separate Historian tags for most DAServers/PLCs, but if you instead map the entire integer to a single Historian tag, you can address individual bits using standard SQL Server queries.

For example, consider the following query that returns process data values for the 'SysTimeMin' tag:

```
SELECT Value FROM dbo.History WHERE TagName = 'SysTimeMin'
```

However to get more bits of data, you can add 2 (bitposition-1) and use bitwise & operator on the Value column as shown in the following query. Using the the Integer cast, you can query a maximum of 32 bits.

```
SELECT
CONVERT(BIT, CAST(Value AS INT) & 1)   As 'Bit0',
CONVERT(BIT, CAST(Value AS INT) & 2)   As 'Bit1',
CONVERT(BIT, CAST(Value AS INT) & 4)   As 'Bit2',
CONVERT(BIT, CAST(Value AS INT) & 8)   As 'Bit3',
CONVERT(BIT, CAST(Value AS INT) & 16)  As 'Bit4',
CONVERT(BIT, CAST(Value AS INT) & 32)  As 'Bit5',
CONVERT(BIT, CAST(Value AS INT) & 64)  As 'Bit6',
CONVERT(BIT, CAST(Value AS INT) & 128) As 'Bit7'
FROM dbo.History WHERE TagName = 'SysTimeMin'
```

The results are:

Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7
0	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0
1	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0
1	0	1	1	0	0	0	0
0	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0

CHAPTER 4

SQL Queries for Alarms and Events

Note: The alarm and event history functionality described in this chapter captures detailed histories from Application Server. This functionality should not be confused with the Classic Event subsystem allows for some basic events tracking and is based on historical data.

AVEVA Historian captures process data about your plant. In addition to real-time and historical data, this includes information about events.

Events are like other process data – for example, temperature – because their values can change over time. Events differ from other process data in these ways:

- Events usually change more slowly.
- Events usually are more complex than simply a value, time, and quality.

Event data answers questions like "When did this setpoint change and who changed it?" The event record could include the name of the operator, the workstation from where the change was made, any comment about the change, the name of the person who verified the change, and other related details.

Alarms are a specific kind of event. They represent state changes and have an associated lifecycle. This lifecycle includes these states (usually in this order):

- Set – For example, when a temperature goes too high.
- Acknowledged – That is, when an operator recognizes it as an alarm and, ideally, addresses it.
- Clear – For example, when the temperature returns to normal.

Alarms may also have other states, but these are the key ones.

You can query the Events view, which references the History table, to track and analyze alarms and other events. Because Events is actually an extension table (like History), its data is stored in blocks, not in SQL Server tables.

Note: The Events view does not expose all application-specific columns that may be stored by Historian. (Such columns are queryable from the REST/OData interface.) Also, it is not unusual for Events columns to contain many NULL values.

For more information about the Events view, see Events in the *AVEVA Historian Database Reference*.

Querying Alarms and Events

Querying Alarm and Event data is helpful in keeping track of your production environment. For example, as an operator, you could query the data to find out answers to these questions:

- How often alarm events occur?
- Where in the facility they occur?
- How critical the alarms are?
- How quickly they are addressed and cleared?

Datetime in Alarm and Event Queries

All queries of alarm and event data must include at least one datetime column.

Alarm and event queries use two ways to express time:

- **UTC time**
  This format is used by EventTimeUtc.
- **Local time for the location of the Historian server**
  This is used by columns like EventTime.

The value of wwTimeZone affects any datetime recorded in local time, but does not affect EventTimeUtc.

Example: Listing all events

One of the simplest queries for alarms and events data is to get a list of all events.

For example,a query like this one would list all events between the dates and times specified:

```
SELECT *
   FROM Events
   WHERE EventTime between '2015-10-25 0:00' and '2015-10-26 0:00'
```

Example: How often alarms occur

This query reports the average alarm rate on hourly basis.

```
DECLARE @StartTime as varchar(60)
DECLARE @EndTime as varchar(60)
SET @StartTime = '2015-10-25 12:00:00'
SET @EndTime = '2015-10-26 12:00:00'

DECLARE @AlarmRaise table
(
       EventTime nvarchar(60),
       ID nvarchar(50),
       AlarmState nvarchar(20),
       SourceArea nvarchar(20),
       SourceObject nvarchar(20)
)
INSERT @AlarmRaise select
EventTime,Alarm_ID,Alarm_State,Source_Area,Source_Object from Events where
EventTime > @StartTime and EventTime < @EndTime and Alarm_State='UNACK_ALM'
--=======================--

DECLARE @AlarmCounts table
(
       ForDate nvarchar(60),
       OnHour nvarchar(50),
       CountperHour nvarchar(20)

)

INSERT @AlarmCounts SELECT CAST(EventTime as date) AS ForDate,
       DATEPART(hour,EventTime) AS OnHour,
       Count(*) AS "CountperHour"
FROM @AlarmRaise
GROUP BY CAST(EventTime as date),
       DATEPART(hour,EventTime)
```

```
SELECT Avg(CAST(CountperHour as INT)) as "Average Alarm Rate on Hourly Basis"
from @AlarmCounts
```

The resulting report looks like this:

Average Alarm Rate on Hourly Basis
6

Example: Most frequent alarm per hour

This query reports the most frequent alarms for each hour:

```
DECLARE @StartTime as varchar(60)
DECLARE @EndTime as varchar(60)
SET @StartTime = '2017-11-10 12:00:00'
SET @EndTime = '2017-11-10 12:10:00'

DECLARE @AlarmRaise table
(
        EventTime nvarchar(60),
        ID nvarchar(50),
        AlarmState nvarchar(20),
        SourceArea nvarchar(20),
        SourceObject nvarchar(20),
        SourceConditionVariable nvarchar(40)
)
INSERT @AlarmRaise select
EventTime,Alarm_ID,Alarm_State,Source_Area,Source_Object,Source_Condition
Variable from Events where EventTime > @StartTime and EventTime < @EndTime
and Alarm_State='UNACK_ALM'
--=======================--
SELECT CAST(EventTime as date) AS ForDate,
        DATEPART(hour,EventTime) AS OnHour,
        Count(*) AS "Count per Hour",
          SourceObject + SourceConditionVariable AS "Alarm Attribute"

FROM @AlarmRaise
GROUP BY CAST(EventTime as date),
        DATEPART(hour,EventTime),
          SourceObject,
          SourceConditionVariable
ORDER BY ForDate ASC,OnHour,[Alarm Attribute]
```

This resulting report looks like this:

ForDate	OnHour	Count per Hour	Alarm Attribute
2017-11-10	12	10	AlarmHeartBeatAlarmHeartBeat.AlmHeartBeat.Hi
2017-11-10	12	2	Reactor_31Reactor_31.ReactLevel.Hi
2017-11-10	12	2	Reactor_31Reactor_31.ReactLevel.Lo
2017-11-10	12	2	Reactor_31Reactor_31.ReactTemp.Hi
2017-11-10	12	1	StorageTank_31StorageTank_31.ProdLevel.Lo

2017-11-10	12	6	VectorTagsVectorTags.VectorX.Hi
2017-11-10	12	3	VectorTagsVectorTags.VectorX.HiHi
2017-11-10	12	6	VectorTagsVectorTags.VectorX.Lo
2017-11-10	12	3	VectorTagsVectorTags.VectorX.LoLo
2017-11-10	12	2	VectorTagsVectorTags.VectorZ.Hi

Example: Pinpointing where alarms occur

This query reports the number of alarms raised from each source by area and by object.

```
DECLARE @StartTime as varchar(60)
DECLARE @EndTime as varchar(60)
SET @StartTime = '2015-10-25 12:00:00'
SET @EndTime = '2015-10-26 12:00:00'

DECLARE @AlarmRaise table
(
	EventTime nvarchar(60),
	ID nvarchar(50),
	AlarmState nvarchar(20),
	SourceArea nvarchar(20),
	SourceObject nvarchar(20)
)
INSERT @AlarmRaise select
EventTime,Alarm_ID,Alarm_State,Source_Area,Source_Object from Events where
EventTime > @StartTime and EventTime < @EndTime and Alarm_State='UNACK_ALM'
--======================--
SELECT SourceArea AS "Source Area/Object" , count(*) AS "Total Number of
Alarms" from @AlarmRaise GROUP BY SourceArea UNION
SELECT SourceObject  AS "Source Area/Object" , count(*) AS "Total Number of
Alarms"  from @AlarmRaise GROUP BY SourceObject;
```

The results look like this:

Source Area/Object	Total Number of Alarms
Area_001	6
UserDefined_001	6

Example: Showing average time to clearing an alarm

This query reports the average time to clear Critical, High, Medium, and Low alarms per hour.

```
DECLARE @start DateTime2
SET @start = '2017-12-11'
DECLARE @end DateTime2
SET @end = '2017-12-12'


-- ack time per severity per hour for high, medium and low
```

```
SELECT DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0) as hour,
    e.Severity,
    avg(Alarm_UnAckDurationMs) as avg_unack,
    count(*) as count
FROM Events e
    WHERE
        e.EventTime < @end
        AND e.EventTime >= @start
           AND e.Severity <=3 -- critical = 1, high = 2 medium = 3 low =
    4
           AND e.Type = 'Alarm.Acknowledged'
GROUP BY
     DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0),
           severity
ORDER BY
     DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0),
           e.severity

-- ack time by user by hour
SELECT DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0) as hour,
    avg(Alarm_UnAckDurationMs) as avg_unack,
    e.User_Name,
    count(*) as count
FROM Events e
    WHERE
        e.EventTime < @end
        AND e.EventTime >= @start
           AND e.Type = 'Alarm.Acknowledged'
GROUP BY
     DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0),
           e.User_Name
ORDER BY
     DATEADD(hour, DATEDIFF(hour, 0, e.EventTime), 0),
           e.User_Name
```

This query results in two reports. The first one looks like this:

hour	Severity	avg_unack	count
2017-12-11 08:00:00.000	2	330949	73
2017-12-11 08:00:00.000	3	13723786	1
2017-12-11 09:00:00.000	2	23524	195
2017-12-11 09:00:00.000	3	4931	1
2017-12-11 10:00:00.000	2	22550	182
2017-12-11 11:00:00.000	2	24552	189
2017-12-11 12:00:00.000	2	22474	189
2017-12-11 13:00:00.000	2	23492	192
…			

The second report looks like this:

hour	avg_unack	User_Name	count
2017-12-11 08:00:00.000	453722	DefaultUser	92
2017-12-11 09:00:00.000	24997	DefaultUser	239
2017-12-11 10:00:00.000	22751	DefaultUser	222
2017-12-11 11:00:00.000	25528	DefaultUser	231
2017-12-11 12:00:00.000	23549	DefaultUser	233
2017-12-11 13:00:00.000	23807	DefaultUser	236
2017-12-11 14:00:00.000	25472	DefaultUser	237
2017-12-11 15:00:00.000	25350	DefaultUser	237
…			

Example: Evaluating response time for alarms

This query reports when an alarm is raised, acknowledged, and cleared. The report lists both times and duration.

```
DECLARE @StartTime as varchar(60)
DECLARE @EndTime as varchar(60)
SET @StartTime = '2017-12-12 12:00:00'
SET @EndTime = '2017-12-12 12:02:00'

DECLARE @AlarmRaise table
(
	EventTime nvarchar(60),
	ID nvarchar(50),
	AlarmState nvarchar(20)
)
INSERT @AlarmRaise
   SELECT EventTime,Alarm_ID,Alarm_State
   FROM Events
	WHERE EventTime > @StartTime and EventTime < @EndTime and
Alarm_State='UNACK_ALM'

DECLARE @AlarmAck table
(
	EventTime nvarchar(60),
	ID nvarchar(50),
	UnAckDuration nvarchar(20)
)
INSERT @AlarmAck
   SELECT EventTime,Alarm_ID,Alarm_UnAckDurationMs
   FROM Events
	WHERE EventTime > @StartTime and EventTime < @EndTime and
Alarm_Acknowledged=1

DECLARE @AlarmClear table
(
	EventTime nvarchar(60),
	ID nvarchar(50),
```

```
      AlarmDuration nvarchar(20)
)
INSERT @AlarmClear
   SELECT EventTime,Alarm_ID,Alarm_DurationMs
   FROM Events
      WHERE EventTime > @StartTime and EventTime < @EndTime and
Type='Alarm.Clear'


--=======================--
SELECT 'Alarm Life - '+ s.ID
            ,CASE
                  WHEN a.EventTime > c.EventTime THEN 'Cleared Before
      Ack'
                  WHEN a.EventTime < c.EventTime THEN 'Acked Before
      Clear'
                  ELSE '-' END as Comment
            ,s.EventTime as AlarmRaised
            ,a.EventTime as AlarmAcked
            ,c.EventTime as AlarmClear
            ,a.UnAckDuration as UnAckDuration
            ,c.AlarmDuration as AlarmDuration

FROM (@AlarmRaise s inner join @AlarmClear c on c.ID=s.ID)
LEFT JOIN @AlarmAck a on a.ID=c.ID and a.EventTime<>c.EventTime
ORDER BY AlarmRaised asc
```

The results look like this:

(No column name)	Comment	AlarmRaised	AlarmAcked	AlarmClear	UnAck Duration	Alarm Duration
Alarm Life - B0718EAE-1301-1D00-46D6-061A5265589F	Cleared Before Ack	2017-12-12 12:00:01.2540000	2017-12-12 12:00:56.5170000	2017-12-12 12:00:13.1430000	55263	11889
Alarm Life - 5FE1D46F-C6D7-D7C6-DB24-A832B0361562	Acked Before Clear	2017-12-12 12:00:40.1220000	2017-12-12 12:00:56.5150000	2017-12-12 12:01:00.1220000	16393	20000
Alarm Life - 39DBF54B-B4ED-6D35-C1DF-19EE53D62211	Cleared Before Ack	2017-12-12 12:00:48.1290000	2017-12-12 12:00:56.9600000	2017-12-12 12:00:52.1240000	8831	3995
Alarm Life - 973B8CEA-DBDA-7B4B-183A-E423B1098C91	Acked Before Clear	2017-12-12 12:00:57.6870000	2017-12-12 12:00:58.7880000	2017-12-12 12:01:03.6230000	1101	5936

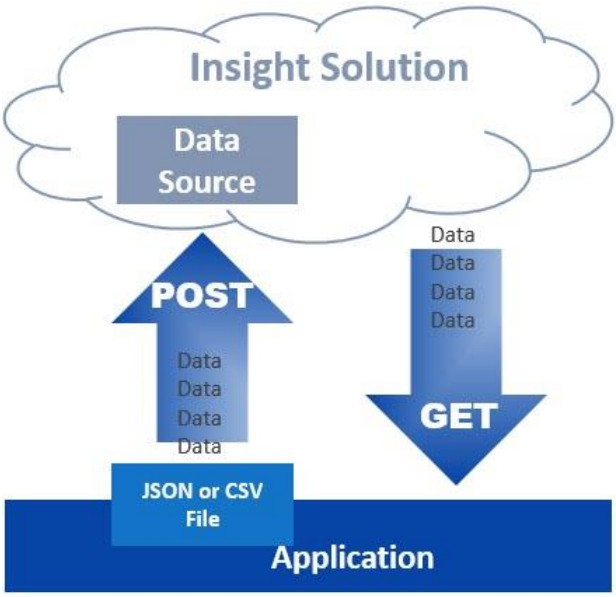
Alarm Life - B926FB71-B6DE-4916-C23C-CC85D5B01BDE	Cleared Before Ack	2017-12-12 12:01:45.7120000	2017-12-12 12:01:57.0140000	2017-12-12 12:01:53.2410000	11302	7529
Alarm Life - A52B65BE-27F2-CC6A-9369-EDFD3E6D7514	Cleared Before Ack	2017-12-12 12:01:47.1270000	2017-12-12 12:01:57.0160000	2017-12-12 12:01:52.7340000	9889	5607
Alarm Life - AFE96385-10B4-F695-9302-320F02FA46A1	Cleared Before Ack	2017-12-12 12:01:51.1570000	2017-12-12 12:01:57.0180000	2017-12-12 12:01:53.6230000	5861	2466

CHAPTER 5

Browser-Friendly Data Retrieval

Historian Data REST API

With Historian Data REST API, you can upload data to and retrieve it from your Insight solution.



You can use these types of requests with Historian Data REST API:

- **Upload requests** use the POST method to an upload endpoint URL.

  Upload requests send data or metadata to a specific data source within your Insight solution.

  See Data upload for details about upload requests.

- **Retrieval requests** use the GET method and a different endpoint URL for retrieval from your Insight solution.

  See *Data retrieval* (see "*Data retrieval*" on page 185) for two methods to retrieve data.

You can submit requests to the Historian Data REST API using a web browser or a client-side applications such as these:

- Microsoft Excel (2013, 2016, or Office 365)
- Business Intelligence (BI) systems, such as Tableau and Microsoft Power BI

Supported OData features

OData is an industry standard for querying and updating data from a variety of sources. The implementation of OData for the Historian Data REST API includes support for:

- JSON and atom formats.
- OData versions 3 and 4.
- Pagination. That is, if your request returns more than 5000 results, they will be returned in pages of up to 5000 records. Each page will include a link to retrieve the next page of records.
- A subset of the OData system query options.

For more information, see OData.org JSON Verbose Format specification.

Recommendation: For best results, when you want to view the data returned by the Historian Data REST API, use the JSONView extension for the Chrome browser.

Supported versions

Insight supports versions 1 and 2 of the Historian Data REST API.

Version 2

This is the current and recommended version of the Historian Data REST API.

Version 2 of this REST APIs based on version 1 and includes further enhancements. Version 2 includes these differences from version 1:

- The `TagFilter` parameter is supported as a part of a GET or POST query used with ProcessValues, AnalogSummary, and StateSummary resources.
- The `datetimeoffset` parameter is not supported as part of the `DateTime` syntax.
- While Raw and ProcessValue entities use `DateTime`, summary entities use `StartDateTime` and `EndDateTime`.
- For most version 2 queries, single quotes are not used for `DateTime`. For example:

  ```
  https://online.wonderware.com/apis/Historian/v2/ProcessValues
  ?$filter=DateTime+gt+2017-07-13T00:00:00
  ```

  However, when querying events, single quotes are required for `DateTime`. For example:

  ```
  https://online.wonderware.com/apis/Historian/v2/Events
  ?$filter=EventTime+gt+'2017-07-13T00:00:00'
  ```

  By contrast, version 1 queries do use single quotes for `DateTime`. For example:

  ```
  https://online.wonderware.com/apis/Historian/v1/ProcessValues
  ?$filter=DateTime+gt+datetimeoffset'2017-07-13T00:00:00'
  ```

- *TagProperty* (see "*TagProperties*" on page 214) and *Events* (see "*Events*" on page 202) entity types are now open type. That is, dynamic properties can be added to the response at runtime. This can be verified using `$metadata` endpoint URL:

  ```
  https://online.wonderware.com/s/<solution_id>/apis/historian/v2/$metadata
  ```

  where there will be additional attribute, `OpenType="true"`, under `<EntityType>` section.

- The *Tags resource* (see "*Tags*" on page 208) returns all the properties (fixed and extended) for a tag. A tag's extended properties will be added to the response only if they exist. (The extended property name will not be listed for tags that do not have a given extended property.)
- The combined *Summary resource* (see "*Summary (v1 only)*" on page 222) is not supported in version 2. Use the individual *AnalogSummary* (see "*AnalogSummary*" on page 193) and *StateSummary* (see "*StateSummary*" on page 199) resources instead to retrieve the summary of a tag.
- Insight returns a list of resources and endpoints in JSON format instead of the previously used XML when you specify the default endpoint URL for your solution (where <*solution_id*> is the identifier for your Insight solution:

  ```
  https://online.wonderware.com/s/<solution_id>/apis/historian/v2
  ```

- Version 2 adds support for the OData `contains` function for applicable resources and properties.

Version 1

This is the original version of Historian Data REST API based on the OData v4 specification.

- Version 1 uses the `DateTime` format used in this example:

```
https://online.wonderware.com/s/ik97r5/apis/Historian/v1
/AnalogSummary?$filter=FQN+eq+'Baytown.tank_level'
+and+StartDateTime+ge+datetimeoffset'2016-05-14T00:00:00.000-07:00'
+and+EndDateTime+le+datetimeoffset'2016-05-16T00:00:00.000-07:00'
```

iHistory and Account Authentication

The iHistory web service requires users to be authenticated via a login process before they can retrieve data. The process for this differs between AVEVA Historian Insight and AVEVA Insight:

- **AVEVA Historian Insight (on-premises)**
  Uses Windows integrated security. A user must belong to the aaAdministrators, aaPowerUsers, or aaUsers Windows group. The iHistory web service uses Negotiate authentication, which is supported by most modern browsers and web service clients (such as Microsoft Excel).
- **AVEVA Insight (cloud-based)**
  Uses OpenID Connect and Basic authentication. Users must be invited to a specific account within AVEVA Insight and can then access all data published to that account. When using OpenID Connect, the iHistory web service uses "bearer token" authentication, per the OpenID Connect standard. As relatively new standard, some web service clients (for example, Microsoft Excel) do not have native support for it. However, some of those same clients (for example, Microsoft Power BI) support Basic authentication.

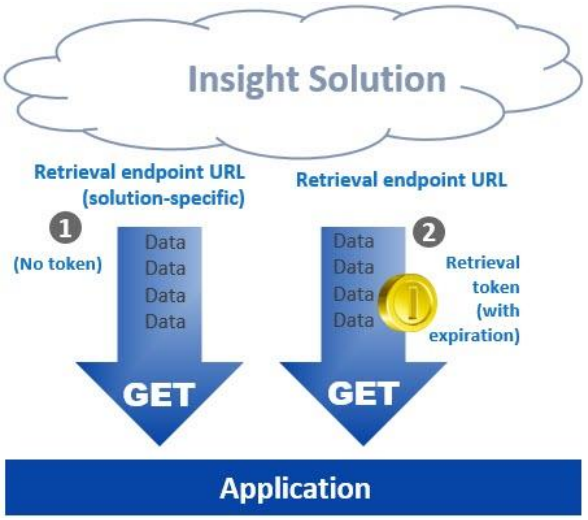
When you query on-premises data via iHistory, you will be prompted for authentication to the Historian.

- If you have a valid Windows login on the historian, you can retrieve general information about the services and the kinds of data available.
- If you belong to the aaAdministrators, aaPowerUsers, or aaUsers Windows group, you can retrieve actual application data -- including events, process history, tag information, and so on.

Data retrieval

The Historian Data REST API allows you to retrieve data from your Insight solution.

Retrieval requests use a GET method and a retrieval endpoint URL. The retrieval endpoint URL differs depending on whether you are using token authentication (with a retrieval token) or basic authentication, with no token.



- **GET method with basic authentication**
  Requests to retrieve data via basic authentication require no retrieval token. Rather, these requests must use the endpoint URL specified by the Integration Settings page. Get details here.
- **GET method with token authentication**
  You can use retrieval token to access your Insight data using token authentication. Get details here.

Forming retrieval requests

- With basic authentication, use this syntax to form your retrieval requests:

```
https://online.wonderware.com/s/<solution_ID>/apis/historian/<api_version
>/<resource>?<query_parameters>
```

- With token authentication, use syntax to form your retrieval requests:

```
https://online.wonderware.com/apis/historian/<api_version>/<resource>?<query_parameters>
```

For more information, see Retrieval endpoints and tokens.

This URL includes these parts:

Syntax element	Explanation
https://online.wonderware.com/	The base URL
s/<*solution_ID*>	The unique identifier for your Insight solution. This is not used if you use token authentication with a solution-specific retrieval token.
apis/historian/<*api_version*>	The API and *version* (see "*Supported versions*" on page 184)
<*resource*>	The Historian Data REST API retrieval resource. *See the complete list* (see "*Retrieval resources*" on page 189).

?<*query_parameters*>	Query parameters.
	These may be OData parameters using OData syntax tokens and operators; for example "$filter" as in this syntax:
	`https://online.wonderware.com/apis/historian/v2/AnalogSummary?$filter=FQN+eq+'Baytown.tank_level'`
	Or, REST parameters; for example "TagFilter", as in this syntax:
	`https://online.wonderware.com/apis/historian/v2/AnalogSummary?TagFilter=FQN eq 'Baytown.tank_level'`
	Note that TagFilter is used only with *AnalogSummary* (see "*AnalogSummary*" on page 193), *ProcessValues* (see "*ProcessValues*" on page 191), and *StateSummary* (see "*StateSummary*" on page 199).

For example, this retrieval request gets analog summary data via an endpoint URL using basic authentication:

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/AnalogSummary
?$filter=FQN+eq+'Depot.Train09'+and+StartDateTime+ge+2017-06-09T09:00:00-
07:00+and
+EndDateTime+ge+2017-06-09T10:00:00-07:00&Resolution=3600000
```

Two powerful parameters: RetrievalMode and Resolution

You can use the RetrievalMode and Resolution parameters in retrieval queries to better control of your search results.

- **RetrievalMode** specifies how the resulting data is calculated for Raw and ProcessValues entities. Valid values are:
  - Average
  - Cyclic
  - Integral
  - Minimum

- BestFit
- Delta
- Interpolated
- Slope
- Counter
- Full
- Maximum

- **Resolution** specifies the granularity of data returned for Raw, ProcessValues, and Summary entities.

Using TagFilter

You can use the TagFilter parameter as a part of a GET or POST query with ProcessValues, AnalogSummary, and StateSummary resources. *See examples.* (see "*Retrieve data using PowerBI*" on page 237)

It allows a query string using OData filter query notation.

TagFilter can include:

- Up to 20 AND clauses
- Up to 20 OR clauses
- Up to 2 UDF clauses

You cannot mix AND and OR in the same query.

Operators should be lowercase.

- Valid Format:
```
Historian/v2/ProcessValues?TagFilter=startswith(Source,'MVDS') and TagType eq 'string'
```
- Invalid Format:
```
Historian/v2/ProcessValues?TagFilter=startswith(Source,'MVDS') And TagType eq 'string'
```

Most searches are case-insensitive. However, search by these attributes is case-sensitive:

- InterpolationType
- MessageOn
- MessageOff

These are the supported (and not supported) features for TagFilter:

Supported	Not Supported

• V2 controllers
• Use with ProcessValues, AnalogSummary and StateSummary
• GET and POST queries
• StartsWith, EndsWith
• SkipToken
• Nested query with same operator type ("and" or "or"). Examples:
`Historian/v2/ProcessValues?TagFilter=startswith(Source,'MVDS') and TagType eq 'string' and EngUnit eq 'None'`
`Historian/v2/ProcessValues?TagFilter=startswith(Source,'MVDS') or TagType eq 'string' or EngUnit eq 'None'`
	• V1 controller
	• Toupper and tolower functions
	• Nested query with grouping precedence operator; that is: '()'
	• Query with mixed operator like "and, or"
	• These attributes support only "eq" and not "startswith or endswith"
	  ○ EngUnitMax
	  ○ EngUnitMin
	  ○ InterpolationType
	  ○ MessageOff
	  ○ MessageOn
	  ○ Any TagExtendedProperty that is not a string (booleans, integers, doubles, guids)
	• Geospatial primitives, such as Geolocation, Geography, Geometry are not fully supported by TagFilter .

Supported syntax tokens and operators

These tokens are supported for system query via the Historian Data REST API. They are all case-sensitive.

Token	Description
$filter	Specifies an expression or function that must evaluate to true for a record to be returned in the collection. All typical OData functions are supported for the $filter clause. The $filter expression supports references to properties and literals. Literal values include: • Strings enclosed in single quotes • Numbers and Boolean values (true or false) Filtering for process value and summary data is case-sensitive. However, while event property names are case-sensitive, filtering is case-insensitive. For example, if you filter property values based on a value of "true," values such as "TRUE," "True," and "true" could be returned. The case returned in the query results reflects the case of the stored value.
$select	Specifies a subset of properties to return.
$skip	Specifies the number of records to skip from the beginning of the result set.

Token	Description
$skiptoken	Used to get the next record set that satisfies the query conditions. You do not need to include this token in the query, but you will see it upon query execution.
$top	Specifies the maximum number of records to return. This subset is formed by selecting only the first *N* items of the set, where *N* is a positive integer specified by this query option.

These logical operators are supported for the query options:

Operator	Description
eq	Equal
ne	Not equal
gt	Greater than
ge	Greater than or equal
lt	Less than
le	Less than or equal
and	Logical and
or	Logical or
not	Logical negation

In the filter expression, you can have only a single time clause combined with a single filter clause using the "and" operator. The filter clause itself can be complex, using any of the supported logical operators. Use parentheses ( ) to create precedence groups within an expression in filter clause.

Note: Use "%20" to indicate a space. Use "%27" to indicate a single quote.
If you are using the JSONView viewer in the Chrome browser, you can use a plus sign (+) to indicate a space to make the URI string more readable.

If the expression includes multiple values for the criteria, you must specify each criteria separately using the "or" operator. For example:

```
...
((Priority+eq+100+or+Priority+eq+200+or+Priority+eq+500+or+Priority+eq+700)+and+(filter …))
```

Retrieval resources

The Historian Data REST API exposes various resources through an endpoint URL that is specific to your Insight solution.

This API includes the following resources for retrieving data:

Process Data and Events Resources	Tag Property Resources	Resources for Version 1 Only
• *ProcessValues* (see "*ProcessValues*" on page 191) • *AnalogSummary* (see "*AnalogSummary*" on page 193) • *StateSummary* (see "*StateSummary*" on page 199) • *Events* (see "*Events*" on page 202)	• *Tags* (see "*Tags*" on page 208) • *TagProperties* (see "*TagProperties*" on page 214) • *TagPropertyValues* (see "*TagPropertyValues*" on page 215) • *TagGroups* (see "*TagGroups*" on page 216) • *TagSuggest* (see "*TagSuggest*" on page 218) • *TagSearch* (see "*TagSearch*" on page 220) • *TagExtendedProperties* (see "*TagExtendedProperties*" on page 221)	• *Summary* • *Daily* (see "*Daily (v1 only)*" on page 223) • *Hourly* (see "*Hourly (v1 only)*" on page 229) • *Minutely* (see "*Minutely (v1 only)*" on page 231)

Note: All property names are case-sensitive. Event storage preserves the case that you provide for any property value. For example, a property value of "TRUE" is different than "True" and "true."

ProcessValues

Description	Retrieves a set of process value records (where each record includes value + time + quality, or VTQ) for the specified tags.
URL	`/ProcessValues`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname. • **DateTime=**[DateTimeOffset] (v2 and later) Specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z.
Optional Parameters	• **OPCQuality =**[Int32] The data quality as reported by the source. • **Value =**[Double] 0 or 1. • **Text=**[String] For string tags, contain the value. For discrete tags, contains the message associated the value (0 or 1).
Success Response	`Code: 200` `Content: { fqn: plant12.pump6, datetime: 2016-09-03T18:44:09.352247Z}`

Error Response	`Code: 404 NOT FOUND`
	`Content: { error : "FQN doesn't exist" }`
	or
	`Code: 401 UNAUTHORIZED`
	`Content: { error : "You are unauthorized to make this request." }`

Sample Query

Scenario 1

This query produces process values for a list of tags that end with "level". In this case, the user doesn't know the fully qualified name (FQN) of a specific tag and wants a short list of possible matches.

```
https://online.wonderware.com/apis/historian/v2/Process
Values
?TagFilter=endswith(FQN, 'level').
```

See more TagFilter examples. (see "*Retrieve data using PowerBI*" on page 237)

Scenario 2

This query returns process values for a specific tag identified by its fully qualified name (*datasource.tagname*). Using a"$filter" clause, it specifies a tag named tank_level within the Baytown data source. The result is a list of values for the tank_level tag.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v
2/ProcessValues
?$filter=FQN+eq+'Baytown.tank_level'
```

Scenario 3

This query includes a start date time, end date time, and other query parameters.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v
2/ProcessValues
?$filter=FQN+eq+'Baytown.tank_level'and DateTime ge
2014-07-04T23:57:29Z
and DateTime le
2014-07-05T00:02:29Z&RetrievalMode=BestFit&Resolution=6
500
```

Sample Output

```
{
    "odata.metadata":
"https://online.wonderware.com/s/ik97r5/apis/historian/
v2/
$metadata#ProcessValues",
    "value": [
        {
            "FQN": "20140805AK.TestSkipToken_0",
            "DateTime": "2014-08-06T17:25:19.216486Z",
            "OpcQuality": 192,
            "Value": 39816,
            "Text": "39816"
        },
        {
            "FQN": "20140805AK.TestSkipToken_0",
            "DateTime": "2014-08-06T17:25:20.196Z",
            "OpcQuality": 192,
            "Value": 39817,
            "Text": "39817"
        }
    ],
}
```

AnalogSummary

Description	Retrieves analog statistics for the specified tags.
URL	`/AnalogSummary`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname. • **StartDateTime=**[DateTimeOffset] The starting date and time. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 1985-04-12T23:20:50.52435Z • **EndDateTime=**[DateTimeOffset] The ending date and time. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 1985-04-12T23:20:50.52435Z
Optional Parameters	• **RetrievalMode=**[string] Possible values are: Cyclic, Full. Default is Cyclic. • **Resolution=**[Int] In milliseconds. Any positive integer. • **SliceBy=**[Int|Discrete|String] Performs dynamic resolution/cycle computation by tags. Returns one Analog Summary value per tag per dynamic cycle with start and end date time. SliceBy can support up to 10 tags. • **SliceByValue=**[string] Specifies the filter criterion to get the summary values for SlicedBy, based on that filter value. • **OPCQuality=**[Int32] Defines the OPC quality for the data. Normal OPC quality retrieval logic is applied if all the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned.

	• **PercentGood=**[Double] The ratio of the number of rows that have "good" quality to the total number of rows in the retrieval cycle, expressed as a percentage in the range 0 to 100. • **First=**[Double] If at least one non-NULL point exists for the tag in question within the retrieval cycle, then the value returned is the first point stored with a time stamp within the retrieval cycle. If no points exist within the retrieval cycle, then the value returned is the current value at the cycle start time. If no non-NULL points can be found, then NULL is returned. • **FirstDateTime=**[DateTimeOffset] Timestamp associated with first value. This might be earlier than StartDateTime if this is the initial value for the retrieval cycle. • **Last=**[Double] If at least one non-NULL point exists for the tag in question within the retrieval cycle, then the value returned is the last point stored with a time stamp within the retrieval cycle. If no points exist within the retrieval cycle, then the value returned is the current value at the cycle start time. If no non-NULL points can be found, then NULL is returned. • **LastDateTime=**[DateTimeOffset] Timestamp associated with last value. This might be earlier than StartDateTime if this is the initial value for the retrieval cycle. • **Minimum=**[Double] If at least one non-NULL point exists for the tag in question within the retrieval cycle, then the value returned is the minimum point stored with a time stamp within the retrieval cycle. If no points exist within the retrieval cycle, then the value returned is the current value at the cycle start time. If no non-NULL points can be found, then NULL is returned. • **MinDateTime=**[DateTimeOffset] Timestamp associated with Min value. NULL if Min is NULL. • **Maximum=**[Double] If at least one non-NULL point exists for the tag in question within the retrieval cycle, then the value returned is the maximum point stored with a time stamp within the retrieval cycle. If no points exist within the retrieval cycle, then the value returned is the current value at the cycle start time. If no non-NULL points can be found, then NULL is returned. • **MaxDateTime=**[DateTimeOffset] Timestamp associated with Max value. NULL if Max is NULL.

	• **Average=**[Double] Time weighted average value of retrieval cycle. This is calculated by using the individual summary averages. The calculation is "Sum(average * delta t) / Total time of average in all cycles" - delta t is prorated for any partially contained storage cycles For analog tags, the calculation is "Sum(value * delta t) / Total time. (This is like the values returned by an Average query against the History table for a cycle of the same length, where the History row DateTime is the same as the EndDateTime here.) • **StdDev=**[Double] Time weighted standard deviation value of the retrieval cycle. The value is calculated using time weighted sums (Integrals) and time weighted sums of squares (IntegralOfSquares) values, prorated for any partially contained storage cycles. For analog tags, similar StdDev values are produced for each cycle. • **Integral=**[Double] Area under value curve of retrieval cycle. The calculation is "Sum(value * delta t) / Total time of integral in all cycles" - delta t is prorated for any partially contained storage cycles For analog tags, the calculation is "Sum(value * delta t) / Total time. (This is like the values returned by an Integral query against the History table for a cycle of the same length, where the History row DateTime is the same as the EndDateTime here.) For analog tags, similar Integral values are produced for each cycle. • **Count=**[Double] Number of values in a particular cycle.

Sample Query	**Scenario 1**
	This query produces a list of tags that end with "level". In this case, the user doesn't know the fully qualified name (FQN) of a specific tag and wants a short list of possible matches.
	`https://online.wonderware.com/apis/historian/v2/AnalogSummary?TagFilter=endswith(FQN, 'level').`
	See more TagFilter examples. (see "*Retrieve data using PowerBI*" on page 237)
	Scenario 2
	This query produces a list of values with analog summary for the tank_level tag.
	Notice that this example uses a fully qualified name ("Baytown.tank_level"), which is a combination of a data source name ("Baytown") and a tagname ("tank_level").
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/AnalogSummary?$filter=FQN+eq+'Baytown.tank_level'`
	Because this query specifies no start or end time and no resolution, these defaults are used for the returned results:
	• EndTime defaults to DateTime.UtcNow
	• StartTime defaults to one hour before EndTime
	• Resolution defaults toTimespan
	• Count defaults to1 (number of returned rows)

Scenario 3

This query produces a list of tag values with analog summary data. The "$filter" clause narrows the results further by specifying these parameters:

- Fully qualified name of the tag (FQN+eq+'Baytown.tank_level')
- Start and end times
- Result set returned at 1 hour intervals (Resolution=3600000). There are 3.6 million milliseconds in an hour.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v
2/AnalogSummary
?$filter=FQN+eq+'Baytown.tank_level'+and+StartDateTime+
ge+2017-05-14T00:00:00.000Z
+and+EndDateTime+le+2017-05-16T00:00:00.000Z&Resolution
=3600000
```

Scenario 4

This query specifies only StartTime and Resolution (600000 ms, or 10 minutes), but no EndTime.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v
2/AnalogSummary
?$filter=FQN+eq+'Baytown.tank_level'
and StartDateTime ge
2017-06-29T00:00:00Z&Resolution=600000
```

In this case, Insight assumes:

- EndTime defaults to DateTime.UtcNow

The number of rows returned (Count) depends on the StartTime and EndTime.

Scenario 5

This query specifies only EndTime, but no StartTime or Resolution.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v
2/AnalogSummary
?$filter=FQN+eq+'Baytown.tank_level'and EndDateTime le
2017-06-29T00:00:00Z'
```

In this case, Insight assumes:

- StartTime defaults to one hour before EndTime
- Resolution defaults toTimespan
- Count defaults to1 (number of returned rows)

	Scenario 6 This query specifies uses SliceBy. `https://online.wonderware.com/apis/Historian/v2/AnalogSummary?$filter=FQN+eq+'Baytown.R21.Level'+and+StartDateTime+ge+2019-03-05T00:00:00.000Z+and+EndDateTime+le+2019-03-05T12:00:00.000Z&SliceBy=Baytown.R21.Batch` In this case, Insight calculates an analog summary for the Baytown..R21.Level tag and batches the results per value for the Baytown.R21.Batch tag.

StateSummary

Description	Retrieves state summary values for the specified tags.
URL	`/StateSummary`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname. • **StartDateTime=**[DateTimeOffset] The starting date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z • **EndDateTime=**[DateTimeOffset] The ending date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z
Optional Parameters	• **RetrievalMode=**[string] Possible values are: Cyclic, Full. Default is Cyclic. • **Resolution=**[Int] In milliseconds. Any positive integer. • **OPCQuality=**[Int32] OPC quality. Normal OPC quality retrieval logic is applied if: All the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned. • **Text=**[string] Non-numeric state. • **Average=**[Double] Average time in this state among all occurrences of this state during this retrieval cycle, including state occurrences that fall only partially within the period. • **AverageContained=**[Double] Average time in this state among all occurrences of this state during this retrieval cycle, excluding state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible.

Optional Parameters	• **Minimum=**[Double] Minimum time in this state among all occurrences of this state during this retrieval cycle, including state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **MinimumContained=**[Double] The minimum of the contained times in this state among all occurrences of this state during the entire retrieval cycle, excluding state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **Maximum=**[Double] Maximum time in this state among all occurrences of this state during this retrieval cycle, including state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **MaximumContained=**[Double] The maximum of the contained times in this state among all occurrences of this state during the entire retrieval cycle, excluding state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **Total=**[Double] Total time in this state during this retrieval cycle, including state occurrences that fall only partially within the period. • **TotalContained=**[Double] Total time in this state during this retrieval cycle, excluding state occurrences that fall only partially within the period. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **Percent=**[Double] Percent of the time during this retrieval cycle that the tag was in this state, including state occurrences that fall only partially within the period. • **PercentContained=**[Double] The precentage of the entire retrieval cycle time that the tag was in this state, excluding state occurrences that fall only partially within the period. This is a ratio between StateTimeTotalContained and StateTimeTotal expressed as a percentage in the range 0 to 100. An occurrence that was partially contained in two or more consecutive storage cycles is converted to a contained state within the retrieval cycle if possible. • **Count=**[Int64] The number of times the state occurred within the retrieval cycle, including states that only partially occur in the cycle. • **CountContained=**[Int64] The number of times the state occurred fully contained within the retrieval cycle. States that only partially occur in the cycle are not counted. • **LastDateTime=**[DateTimeOffset] The timestamp for the last received value.

Sample Query	**Scenario 1**
	This query produces a list of tags that end with "pump_03". In this case, the user doesn't know the fully qualified name (FQN) of a specific tag and wants a short list of possible matches.
	`https://online.wonderware.com/apis/historian/v2/StateSummary?TagFilter=endswith(FQN, 'pump_03').`
	See more TagFilter examples. (see "*Retrieve data using PowerBI*" on page 237)
	Scenario 2
	This query specifies no start or end time and no resolution.
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/StateSummary?$filter=FQN eq 'Baytown.pump_03'`
	In this case, these defaults are used for the returned results:
	• EndTime defaults to DateTime.UtcNow
	• StartTime defaults to one hour before EndTime
	• Resolution defaults toTimespan
	• Count defaults to1 (number of returned rows)
	Scenario 3
	This query specifies only StartTime and Resolution (600000ms, or 10 minutes), but no EndTime.
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/StateSummary?$filter=FQN eq 'Baytown.pump_03' and StartDateTime ge 2017-06-29T00:00:00Z&Resolution=600000`
	In this case, Insight assumes:
	• EndTime defaults to DateTime.UtcNow
	The number of rows returned (Count) depends on the StartTime and EndTime.
	Scenario 4
	This query specifies only EndTime, but no StartTime or Resolution.
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/StateSummary?$filter=FQN eq 'Baytown.pump_03' and EndDateTime le 2017-06-29T00:00:00Z`
	In this case, Insight assumes:
	• StartTime defaults to one hour before EndTime
	• Resolution defaults toTimespan
	• Count defaults to1 (number of returned rows)

Events

Description	Retrieves information about events and alarms. **Note:** All property names are case-sensitive. Event storage preserves the case that you provide for any property value. For example, a property value of "TRUE" is different than "True" and "true."
URL	`/Events`
Method	`GET`
Optional Parameters	• **ID**=[GUID] Globally unique identifier for the event. • **EventTime**=[DateTime] UTC timestamp indicating when the event occurred. • **Type**=[String] Main categorization of the event. Examples of valid values: - Alarm.Acknowledge - Application.Write - Alarm.Clear - User.Write - Alarm.Set - User.Write.Secured - Alarm.Write - User.Write.Verified • **Priority**=[Int32] Value indicating the importance of the event. Values range from 1 to 999, with lower numbers indicating higher importance. • **Namespace** =[String] The namespace for this event tag. • **Severity**=[Int32] Categorization of the urgency of the event: 1 - Critical 2 - Major 3 - Minor 4 - Informational • **EventTimeUTCOffsetMins**=[Int32] For local time, the offset in minutes from UTC time. • **ReceivedTime**=[DateTime] UTC timestamp indicating when the event was received by the Historian server. • **IsAlarm**=[Bool] "true" or "false" indicating whether this event is an alarm. • **Comment**=[String] A comment providing more information about the event. **Note:** If an alarm comment from Application Server contains a backslash (for example, \n), then an extra backslash appears in the event result (for example, \\n) in the browser.

Optional Parameters	• **InTouchType**=[String] InTouch Type value. Examples include: - ALM - RTN - ACK - SYS • **ValueString**=[String] The current value string. • **PreviousValueString**=[String] The previous value string.

Optional Alarm Parameters	• **Alarm_ID**=[String] ID of the original Alarm event. For "Alarm.Set" events, this will be the same as the event ID. • **Alarm_Class**=[String] InTouch alarm classification. Examples include "DSC" (discrete), "VALUE" (LoLo, Hi, etc.), "DEV" (deviation), and "ROC" (rate of change). • **Alarm_Type**=[String] InTouch alarm type. • **Alarm_InAlarm**=[Boolean] "true" or "false" indicating whether the Alarm is still in the active state. • **Alarm_Acknowledged**=[Boolean] "true" or "false" indicating whether the user has acknowledged this alarm. • **Alarm_Condition**=[String] The condition being alarmed. Examples include "Limit.Hi", "ROC.Lo", and "System" among others. • **Alarm_ValueString**=[String] Value logged for the variable related to the event. • **Alarm_LimitString**=[String] Limit being alarmed. • **Alarm_UnAckDuration**=[Int32] The duration, in milliseconds, for which the alarm went un-acknowledged. • **Alarm_Duration**=[Int32] The duration, in milliseconds, for which the alarm was active. • **Alarm_IsSilenced**=[Boolean] "true" or "false" indicating whether the alarm was silenced. • **Alarm_IsShelved**=[Boolean] "true" or "false" indicating whether the alarm was shelved. • **Alarm_ShelveStartTimeUTC**=[DateTime] Scheduled start of the shelve time if the alarm has been shelved. • **Alarm_ShelveEndTimeUTC**=[DateTime] Scheduled end of the shelve time if the alarm has been shelved. • **Alarm_ShelveReason**=[String] The reason the alarm was shelved. • **Alarm_ShelveUserLogin**=[String] The login ID for the person who shelved the alarm. • **Alarm_ShelveNode**=[String] The node of the shelved alarm.

Optional Provider Parameters	• **Provider_NodeName**=[String] Name of the node that generated the event. • **Provider_System**=[String] Software system that generated the event. Examples include "Application Server", "InTouch", and "InBatch". • **Provider_ApplicationName**=[String] Application that generated the event. For Application Server, this will be the galaxy name. For InTouch, it will be the InTouch application name. • **Provider_SystemVersion**=[String] Software version (for example, 4566.1210.5811.1) for the component identified by Provider_ApplicationName. • **Provider_InstanceNames**=[String] Provider-specific string that uniquely identifies the instance an application on a given node.
Optional Source Parameters	• **Source_Name**=[String] The name of the data source for this event tag. • **Source_ProcessVariable**=[String] Process variable to which the event is related. • **Source_ProcessVariable_Units**=[String] Engineering units used for the process variable. • **Source_ConditionVariable**=[String] Condition variable related to the event. For example, if "TIC101" has a field attribute "PV" and this is a "Hi" alarm, this value will be "TIC101.PV.Hi". • **Source_Object**=[String] Non-hierarchical name for the object to which the event is related, for example, "TIC101". • **Source_HierarchicalObject**=[String] Hierarchical name for the source object. For example, "Reactor_001.TIC". • **Source_Area**=[String] Non-hierarchical Area name. For example, "Bottling_Zone". • **Source_HierarchicalArea**=[String] Hierarchical Area name. For example, "Plant_001.Building_002.Mixing". • **Source_Engine**=[String] Non-hierarchical engine name. For example, "AppEngine_001". • **Source_Platform**=[String] Non-hierarchical platform name. For example "WinPlatform_001".

Optional User Parameters	• **User_Account**=[String] This is the login name for the operator for the given application. • **User_Name**=[String] Name of the user. • **User_NodeName**=[String] Computer name from which a user action was executed. • **User_Email**=[String] The user's email address. • **User_Phone**=[String] The user's phone number. • **User_InstanceName**=[String] An instance name for the user. • **User_Agent**=[String] Application name that the user was running when the event was generated.
Optional Verifier Parameters	• **Verifier_Account**=[String] This is the login name for theverifier. • **Verifier_Name**=[String] Name of the verifier. • **Verifier_NodeName**=[String] Computer name from which a verifier action was executed.

Sample Queries	`https://online.wonderware.com/s/ik97r5/apis/Historian/v2/Events ?$filter=EventTime+gt+'2017-07-13T00:00:00'`
	
	`https://online.wonderware.com/s/ik97r5/apis/Historian/v2/Events ?$filter=type+eq+'Alarm.Clear'`

Sample Output


```
{
@odata.context:
"https://online.wonderware.com/s/ik97r5/apis/Historian/v2/
Events
/$metadata#Events",
value: [
{
id: "ee9b042c-181e-4872-b4e8-cb9575c1f4f9",
eventtime: "2017-07-13T00:00:01.047Z",
type: "Alarm.Clear",
receivedtime: "2017-07-13T00:00:01.2553666Z",
source_name: "AaEvents",
namespace: "AaEvents",
alarm_acknowledged: false,
alarm_class: "VALUE",
alarm_condition: "Limit.LoLo",
alarm_durationms: 5006,
alarm_id: "f1e21e43-352e-d3db-7552-6e77f193d02d",
alarm_inalarm: false,
alarm_isshelved: false,
alarm_issilenced: false,
alarm_limitstring: "10.0",
alarm_originationtime: "2017-07-12T23:59:56.041Z",
alarm_state: "UNACK_RTN",
alarm_tagtype: "S",
alarm_type: "LoLo",
comment: "Severity 1",
eventtimeutcoffsetmins: -420,
intouchtype: "LoLo",
isalarm: true,
priority: 1,
provider_applicationname: "AlarmsandEvents",
provider_nodename: "INSIGHTCHARTAPP",
provider_system: "Application Server",
provider_systemversion: "4966.1210.14578.2",
severity: 1,
source_area: "Plant_Area",
source_conditionvariable:
"Drum_Conveyor.HorizontalMovement.LoLo",
source_hierarchicalarea:
"Enterprise.Site.Plant.Plant_Area",
source_hierarchicalobject: "Drum_Conveyor",
source_object: "Drum_Conveyor",
source_processvariable: "Drum_Conveyor.HorizontalMovement",
valuestring: "30.0"
},
```

Tags

Description	Using the GET verb, retrieves tag metadata. For version 2, this also includes tag extended properties. Using the POSTverb, manages multiple tags at once. Using the DELETE verb, deletes a specified tag.
URL	`/Tags`

Methods	`GET`
	`POST`
	`DELETE`

Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname.
Optional Parameters	• **Source=**[string] The data source. • **Description=**[string] The description of the tag. • **EngUnit=**[string] The engineering units used for the tag's recorded values. **EngUnitMax=**[Double] The maximum value of the tag, measured in engineering units. • **EngUnitMin=**[Double] The minimum value of the tag, measured in engineering units. The minimum value of the tag, measured in engineering units. • **InterpolationType=**[string] The interpolation type for retrieval. 0 = Stair-stepped interpolation; 1 = Linear interpolation (if applicable, based on the tag type); 254 = System default interpolation mode. The system default interpolation type is to use the system default for the analog type, either integer or real. The system default interpolation type for an analog type is determined by the setting of the InterpolationTypeInteger and InterpolationTypeReal system parameters. This setting impacts Interpolated, Average, and Integral retrieval modes. • **IntegralDivisor=**[Double] The factor to be applied when integrating a rate with the units [EngUnits/TimeUnit] to a quantity with units [EngUnits]. This factor is called the integral divisor. The default value of 1 assumes a time unit of seconds and ensures that a rate of [Unit/second] is correctly integrated to [Unit]. For a time unit of minutes, set the integral divisor value to 60; for a unit of hours, set the integral divisor value to 3600. The integral divisor is applied similarly to rates or quantities that are not expressed in terms of a time unit. For example, to convert watts to watt-hours, the integral divisor is 1/3600. To convert watts to kilowatt-hours, the integral divisor is 1/3600000. Internal use only.

Optional Parameters	• **RolloverValue=**[Double] The rollover value for the tag. • **MessageOff=**[string] The message associated with the FALSE state of the discrete tag. The maximum number of characters is 64. A discrete tag set to 0 is in the FALSE state. • **MessageOn=**[string] The message associated with the TRUE state of the discrete tag. The maximum number of characters is 64. A discrete tag set to 1 is in the TRUE state. • **TagName=**[string] The unique name of the tag within the AVEVA Historian system. • **TagType=**[string] The type of tag. 1 = Analog; 2 = Discrete; 3 = String; 5 = Event, 7 = Summary tag (analog or state). TagType is a foreign key from the AVEVA Historian TagRef table. • **Raw** Identifies related the raw acquired value. • **Minutely** Identifies related minutely summary. • **Hourly** Identifies related hourly summary. • **Daily** Identifies related daily summary.

Get tags

Sample Query

Scenario 1

This query lists tag metadata for all tags in all data sources for your Insight solution.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/Tags
```

Scenario 2

This query lists metadata for tags in a particular data source. The data source in this example is named Baytown.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/Tags?$filter=Source+eq+'Baytown'
```

Sample Output


```
{
@odata.context:
"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
/$metadata#Tags",
value: [
{
  FQN: "14th aug app.Auto",
  Source: "14th aug app",
  Description: "Automatic mode",
  EngUnit: "",
  EngUnitMax: 0,
  EngUnitMin: 0,
  InterpolationType: "None",
  MessageOff: "Manu",
  MessageOn: "Auto",
  TagName: "Auto",
  TagType: "Discrete",
  Alias: "nareshtag-discrete",
  Location: "/Application Tag 2",
  Raw@odata.navigationLink:

"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
     /Tags('14th%252baug%252bapp.Auto')/Raw",
  Minutely@odata.navigationLink:

"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
     /Tags('14th%252baug%252bapp.Auto')/Minutely",
  Hourly@odata.navigationLink:
"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
/Tags('14th%252baug%252bapp.Auto')/Hourly",
  Daily@odata.navigationLink:

"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
     /Tags('14th%252baug%252bapp.Auto')/Daily"
}
],
@odata.nextLink:
"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
/Tags
?$skiptoken=koen66nr4nGCReC02FqdxO4Y62C7zBVsv3uf7mO8XMQqx
Zveqw8+v2/bVCsFdevT"
}
```

Delete a tag

Sample Query	This query deletes a tag named Weather.Brisbane.Cloudiness.
	`DELETE /Historian/v2/Tags('Weather.Brisbane.Cloudiness')`
	`HTTP/1.1`
	`Host: nchdevruntime.cloudapp.net:8080`
	`Content-Type: application/json`
	`{`
	`"delete":`
	`{`
	`"FQN":["datasourcename.tagname"]`
	`}`
	`}`

Manage multiple tags

Sample Query

Scenario 1: Bulk query

This query retrieves multiple tags.

```
POST /Historian/v2/Tags HTTP/1.1
Host: nchqaruntime.cloudapp.net:8080
Authorization: Basic …….
Content-Type: application/json
Cache-Control: no-cache
{
      "query":
      {

      "FQN":['Weather.Brisbane.Cloudiness','Weather.Auck
land.Cloudiness'],
            "select":"FQN,TagName"
      }
}
```

Scenario 2: Bulk edit

This makes updates to multiple tags. You can edit up to 100 tags at a time, with maximum of 50 properties per tag.

Note: Historian Data REST API extended properties map to the AVEVA Historian SDK tag extended properties API.

This makes updates to a tag's standard and extended properties.

```
POST /Historian/v2/Tags HTTP/1.1
Host: nchdevruntime.cloudapp.net:8080
Authorization: Basic …
Content-Type: application/json
Cache-Control: no-cache

{
      "properties":
      [
            {
                "FQN": "18Jul_kc.test1",
                "PropertyName": "Alias",
                "Text": "SysTimeSec",
                "Type": "String",
                "Searchable":false
            }
}
```

Sample Query	**Scenario 3: Bulk delete**
	This deletes multiple tags.
	`POST /Historian/v2/Tags HTTP/1.1`
	`Host: nchdevruntime.cloudapp.net:8080`
	`Content-Type: application/json`
	`Cache-Control: no-cache`
	`{`
	`"delete":`
	`{`
	`"FQN":["datasourcename.tagname1", "datasourcename.tagname2"]`
	`}`
	`}`

Sample Output	The output from a bulk edit includes results per tag. If any of a tag's properties is invalid, the tag fails and results in an error message.
	`{`
	`"value":`
	`[`
	`{`
	`"FQN": "System.Tag1",`
	`"$StatusCode": 200`
	`},`
	`{`
	`"FQN": "System.Tag2",`
	`"$StatusCode":500,`
	`"$ErrorMessage":"Alias property contains illegal characters"`
	`}`
	`]`
	`}`

TagProperties

Description	Lists all properties used in the system. Tag properties include both base and extended properties of a tag. Examples of TagProperties are DataSourceName and Unit.
URL	`/TagProperties`
Method	`GET`
Required Parameters	• **Name=**[string] The unique name of the tag property.
Optional Parameters	• **Values** Valid values for this property.
Sample Query	This query lists all properties and extended properties for your Insight solution: `https://online.wonderware.com/s/ik97r5/apis/Historian/v2/TagProperties`
Sample Output	{ @odata.context: "https://online.wonderware.com/s/ik97r5/apis/Historian/v2/$metadata#TagProperties", value: [{ Name: "DataSourceName", Type: "String", ReadOnly: true }, { Name: "Namespace", Type: "String", ReadOnly: true }, { Name: "TagName", Type: "String", ReadOnly: true }, { Name: "FullyQualifiedName", Type: "String", ReadOnly: true }, ...

TagPropertyValues

Description	Retrieves values for one or more tag properties. Tag properties include base and extended properties of Tag. Examples of TagProperties are DataSourceName and Unit.
URL	`/TagpropertyValues`
Method	`GET`
Required Parameters	• **Name=**[string] Unique name of the tag property value. • **Value=**[string]
Optional Parameters	• **Tags** Identifies related Tags.
Sample Query	This query lists the property name ("Source") and value (that is, the data source name) for all data sources: `https://online.wonderware.com/s/ik97r5/apis/Historian/v2/TagPropertyValues`

Sample Output	{
	@odata.context:
	"https://online.wonderware.com/s/ik97r5/apis/Historian/v2
	/$metadata#TagPropertyValues",
	value: [
	{
	Name: "Source",
	Value: "Meter",
	Tags@odata.navigationLink:
	"https://online.wonderware.com/s/ik97r5/apis
	/Historian/v2/TagPropertyValues(Name='Source',Value='Meter')
	/Tags"
	},
	{
	Name: "Source",
	Value: "Weather",
	Tags@odata.navigationLink:
	"https://online.wonderware.com/s/ik97r5/apis
	/Historian/v2/TagPropertyValues(Name='Source',Value='Weather')
	/Tags"
	}
	]
	}

TagGroups

Description	Retrieves information about data sources. Each data source contains a group of tags.
URL	`/TagGroups`
Method	`GET`
Required Parameters	• **GroupID=**[string] Globally unique identifier for the tag group.
Optional Parameters	• **GroupName=**[string] The name of the data source. • **TypeID=**[string] The type of data source. • **ParentID=**[string] • **Scope=**[string] Used for tag-level security, this defines a location within the data source. • **Tags** Identifies related Tags. • **Groups** Identifies related TagGroups.
Sample Query	This query lists the data sources (labeled "TagGroups" in this API) for your Insight solution identified by solution ID. In this example, the solution ID used is "ik97r5". `https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagGroups`

Sample Output


```
{
@odata.context:
"https://online.wonderware.com/s/ik97r5/apis/Historian/v
2
/$metadata#TagPropertyValues",
value: [
  {
    Name: "Source",
    Value: "Meter",
    Tags@odata.navigationLink:
    "https://online.wonderware.com/s/ik97r5/apis

/Historian/v2/TagPropertyValues(Name='Source',Value='Met
er')
    /Tags"
  },
  {
    GroupID: "1cd29ff2-f7f3-428b-a593-55c917136a39",
    GroupName: "Weather",
    Type: "1000000",
    ParentID: "0",
    Scope: "Public",
    Tags@odata.navigationLink:
"https://online.wonderware.com
    /s/ik97r5/apis/Historian/v2/TagGroups
    ('1cd29ff2-f7f3-428b-a593-55c917136a39')/Tags",
    Groups@odata.navigationLink:
"https://online.wonderware.com
    /s/ik97r5/apis/Historian/v2/TagGroups
    ('1cd29ff2-f7f3-428b-a593-55c917136a39')/Groups"
  }
]
}
```

TagSuggest

Description	Retrieves content or tags based on search criteria from AVEVA Insight. It returns the search results based on tag name, content name, keywords, description, and so on. It also returns the summary of the search result -- for example, total matching search results for the given search text. You can use the these query options: • q -- (query) Limits the search to only specified fields instead of searching all fields. • Type -- Limits the search to only a certain type of data (Tag or SavedContent).
URL	`/TagSuggest`
Method	`GET`
Required Parameters	• **Value=**[string] The suggested value.
Optional Parameters	• **FieldName=**[string] The related field name. • **Count=**[Double] The hit count for this query string. • **SearchRanking=**[Double] The search ranking for the tag. • **DisplayText=**[string] The associated display text. • **Search** • **Tags** Identifies related Tags. • **ExpandSuggest**

Sample Query	**Scenario 1**
	This query returns values from a search of all fields (default).
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagSuggest`
	Scenario 2
	This query uses the "q" query option to limit the search to only the "location" field.
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagSuggest?q=&searchfields=location`
	Scenario 3
	This query uses the "Type" option to limit the search to only a certain type of data (Tag or SavedContent).
	`https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagSuggest?q=d&Type=Tag`

TagSearch

Description	Provides tagname results based on provided search parameters. The data returned by this entity is dependent on the historian's implementation of search functionality. Some historians will return empty result sets. It finds all the matching results for a given query string. It is basically used to filter the results based on the suggestion results. It searches for the matching record only on the fields provided in the key name as part of the query parameter. The results contain basic tag metadata information; such as FQN, tagname, source, and search ranking. You can use these query options: • q - Specifies the query string (for example, the value typed by the user into the search box). • kn - Specifies the key name (field name) to which the search will be applied. • kv - Specifies the key value which will be used for the search.
URL	`/TagSearch`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname.
Optional Parameters	• **Source=**[string] The data source. • **TagName=**[string] The name of the tag. • **DisplayText=**[string] The displayed text for the tag. • **SearchRanking=**[Double] The search ranking for this tag. • **Tag** Returns a URL to retrieve a list of Tag entity values for the tags that match.
Sample Query	This example uses the "q", "kn", and "kv" query options. `https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagSearch?q=r&kn=source&kv=atron`

TagExtendedProperties

Description	Retrieves both standard and extended properties for a tag.
URL	`/TagExtendedProperties`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified name uses the format: datasource.tagname. • **PropertyName=**[string] The name of the property.
Optional Parameters	• **Value=**[Double] The value of the property. • **Text=**[string] The associated text for the property. • **Type=**[string] Can be one of these: ○ [Edm.]String ○ [Edm.]Int16, Int32, Int64 ○ [Edm.]Double ○ [Edm.]DateTimeOffset ○ [Edm.]Guid ○ [Edm.]Boolean ○ [Edm.]Geography types ○ [Edm.]Geometry types

Summary (v1 only)

Description	Retrieves summary data for a user-defined interval. The additional query option "Resolution" is used to specify the interval to be used for this entity. The earliest and latest requested values of StartDateTime and/or EndDateTime are used for determining the span of time covered by the summary intervals returned.
URL	`/Summary`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified tagname uses the format: DataSourceName.TagName. • **StartDateTime=**[DateTimeOffset] The starting date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z • **EndDateTime=**[DateTimeOffset] The ending date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z
Optional Parameters	• **OPCQuality=**[Int32] OPC quality. Normal OPC quality retrieval logic is applied if: All the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned. • **PercentGood=**[Double] The ratio of the number of rows that have "good" quality to the total number of rows in the retrieval cycle, expressed as a percentage in the range 0 to 100. • **Analog** An embedded collection that maps to *AnalogSummary* (see "*AnalogSummary*" on page 193). • **Split** An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ValueState retrieval mode. • **Contained** An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ContainedState retrieval mode. • **Raw** A URL used to retrieve all stored values for a specified time period and tag. This is equivalent to a preconfigured *ProcessValue* (see "*ProcessValues*" on page 191) query.

Daily (v1 only)

Description	Retrieves daily (24-hour resolution) summary values for the tags specified. The default for Daily is to report summary values for the last week.
URL	`/Daily`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified tagname uses the format: DataSourceName.TagName. • **StartDateTime=**[DateTimeOffset] The starting date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z • **EndDateTime=**[DateTimeOffset] The ending date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z
Optional Parameters	• **TimeZone=**[string] The timezone used for the timeframe. See "Accepted values for TimeZone" below. • **OPCQuality=**[Int32] OPC quality. Normal OPC quality retrieval logic is applied if: All the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned. • **PercentGood=**[Double] The ratio of the number of rows that have "good" quality to the total number of rows in the retrieval cycle, expressed as a percentage in the range 0 to 100. • **Analog=**[analog statistics] An embedded collection that maps to *AnalogSummary* (see "*AnalogSummary*" on page 193). • **Split=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ValueState retrieval mode. • **Contained=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ContainedState retrieval mode. • **Raw** A URL used to retrieve all stored values for a specified time period and tag. This is equivalent to a preconfigured *ProcessValue* (see "*ProcessValues*" on page 191) query. • **Minute**

	• **Hourly**

Accepted values for TimeZone

UTC offset in minutes	TimeZone value
0	UTC
0	Morocco Standard Time
0	GMT Standard Time
0	Greenwich Standard Time
60	W. Europe Standard Time
60	Central Europe Standard Time
60	Romance Standard Time
60	Central European Standard Time
60	W. Central Africa Standard Time
60	Namibia Standard Time
120	Jordan Standard Time
120	GTB Standard Time
120	Middle East Standard Time
120	Egypt Standard Time
120	E. Europe Standard Time
120	Syria Standard Time
120	West Bank Standard Time
120	South Africa Standard Time
120	FLE Standard Time
120	Israel Standard Time
120	Kaliningrad Standard Time
120	Libya Standard Time
180	Arabic Standard Time
180	Turkey Standard Time
180	Arab Standard Time
180	Belarus Standard Time
180	Russian Standard Time

180	E. Africa Standard Time
210	Iran Standard Time
240	Arabian Standard Time
240	Astrakhan Standard Time
240	Azerbaijan Standard Time
240	Russia Time Zone 3
240	Mauritius Standard Time
240	Georgian Standard Time
240	Caucasus Standard Time
270	Afghanistan Standard Time
300	West Asia Standard Time
300	Ekaterinburg Standard Time
300	Pakistan Standard Time
330	India Standard Time
330	Sri Lanka Standard Time
345	Nepal Standard Time
360	Central Asia Standard Time
360	Bangladesh Standard Time
360	Omsk Standard Time
390	Myanmar Standard Time
420	SE Asia Standard Time
420	Altai Standard Time
420	W. Mongolia Standard Time
420	North Asia Standard Time
420	N. Central Asia Standard Time
420	Tomsk Standard Time
480	China Standard Time
480	North Asia East Standard Time
480	Singapore Standard Time
480	W. Australia Standard Time
480	Taipei Standard Time
480	Ulaanbaatar Standard Time

510	North Korea Standard Time
525	Aus Central W. Standard Time
540	Transbaikal Standard Time
540	Tokyo Standard Time
540	Korea Standard Time
540	Yakutsk Standard Time
570	Cen. Australia Standard Time
570	AUS Central Standard Time
600	E. Australia Standard Time
600	AUS Eastern Standard Time
600	West Pacific Standard Time
600	Tasmania Standard Time
600	Vladivostok Standard Time
630	Lord Howe Standard Time
660	Bougainville Standard Time
660	Russia Time Zone 10
660	Magadan Standard Time
660	Norfolk Standard Time
660	Sakhalin Standard Time
660	Central Pacific Standard Time
720	Russia Time Zone 11
720	New Zealand Standard Time
720	UTC+12
720	Fiji Standard Time
720	Kamchatka Standard Time
765	Chatham Islands Standard Time
780	Tonga Standard Time
780	Samoa Standard Time
840	Line Islands Standard Time
-60	Azores Standard Time
-60	Cape Verde Standard Time
-120	UTC-02

-120	Mid-Atlantic Standard Time
-180	Tocantins Standard Time
-180	E. South America Standard Time
-180	SA Eastern Standard Time
-180	Argentina Standard Time
-180	Greenland Standard Time
-180	Montevideo Standard Time
-180	Saint Pierre Standard Time
-180	Bahia Standard Time
-210	Newfoundland Standard Time
-240	Paraguay Standard Time
-240	Atlantic Standard Time
-240	Venezuela Standard Time
-240	Central Brazilian Standard Time
-240	SA Western Standard Time
-240	Pacific SA Standard Time
-240	Turks And Caicos Standard Time
-300	SA Pacific Standard Time
-300	Eastern Standard Time (Mexico)
-300	Eastern Standard Time
-300	Haiti Standard Time
-300	Cuba Standard Time
-300	US Eastern Standard Time
-360	Central America Standard Time
-360	Central Standard Time
-360	Easter Island Standard Time
-360	Central Standard Time (Mexico)
-360	Canada Central Standard Time
-420	US Mountain Standard Time
-420	Mountain Standard Time (Mexico)
-420	Mountain Standard Time
-480	Pacific Standard Time (Mexico)

-480	UTC-08
-480	Pacific Standard Time
-540	Alaskan Standard Time
-540	UTC-09
-570	Marquesas Standard Time
-600	Aleutian Standard Time
-600	Hawaiian Standard Time
-660	UTC-11
-720	Dateline Standard Time

Hourly (v1 only)

Description	Retrieves hourly summary values for the tags specified. The default for Hourly is to report summary values for the last 24 hours.
URL	`/Hourly`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified tagname uses the format: DataSourceName.TagName. • **StartDateTime=**[DateTimeOffset] The starting date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z • **EndDateTime=**[DateTimeOffset] The ending date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z
Optional Parameters	• **TimeZone=**[string] The timezone used for the timeframe. See "Accepted values for TimeZone" below. • **OPCQuality=**[Int32] OPC quality. Normal OPC quality retrieval logic is applied if: All the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned. • **PercentGood=**[Double] The ratio of the number of rows that have "good" quality to the total number of rows in the retrieval cycle, expressed as a percentage in the range 0 to 100. • **Analog=**[analog statistics] An embedded collection that maps to *AnalogSummary* (see "*AnalogSummary*" on page 193). • **Split=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ValueState retrieval mode. • **Contained=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ContainedState retrieval mode. • **Raw** A URL used to retrieve all stored values for a specified time period and tag. This is equivalent to a preconfigured *ProcessValue* (see "*ProcessValues*" on page 191) query. • **Minute**

Minutely (v1 only)

Description	Retrieves summary values with a 1-minute resolution for the tags specified. Default duration for minutely summaries is the last 1 hour. The default for Minutely is to report summary values for the last hour.
URL	`/Minutely`
Method	`GET`
Required Parameters	• **FQN=**[string] The fully qualified name for the tag. A fully qualified tagname uses the format: DataSourceName.TagName. • **StartDateTime=**[DateTimeOffset] The starting date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z • **EndDateTime=**[DateTimeOffset] The ending date and time for the retrieval cycle. This is always specified in UTC using the RFC3339 / ISO8601 format with the Z time zone designator. For example: 2016-09-03T18:44:09.352247Z
Optional Parameters	• **TimeZone=**[string] The timezone used for the timeframe. See "Accepted values for TimeZone" below. • **OPCQuality=**[Int32] OPC quality. Normal OPC quality retrieval logic is applied if: All the point found and processed for this row have GOOD quality. If they all have the same GOOD quality, then that quality is returned. If there is a gap in the entire calculation cycle, then BAD quality is returned for the tag. For any other scenario with any mixture of GOOD and BAD points, a DOUBTFUL OPC quality (64) is returned. • **PercentGood=**[Double] The ratio of the number of rows that have "good" quality to the total number of rows in the retrieval cycle, expressed as a percentage in the range 0 to 100. • **Analog=**[analog statistics] An embedded collection that maps to *AnalogSummary* (see "*AnalogSummary*" on page 193). • **Split=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ValueState retrieval mode. • **Contained=**[collection of state statistics] An embedded collection that maps to *StateSummary* (see "*StateSummary*" on page 199) for the ContainedState retrieval mode. • **Raw** A URL used to retrieve all stored values for a specified time period and tag. This is equivalent to a preconfigured *ProcessValue* (see "*ProcessValues*" on page 191) query.

SystemParameters (on-premises only)

Description	Retrieves names and values for system parameters. (On-premises AVEVA Historian Insight only)
URL	`/SystemParameters`
Method	`GET`
Required Parameters	• **Name=**[string] The unique name for the system parameter.
Optional Parameters	• **Value=**[string] The value of this system parameter. • **Tags** Identifies related tags.

Retrieval examples

You can send retrieval requests to the Historian Data REST API from any modern web browser and from OData client-side tools. Examples of such tools include Microsoft Excel (2013, 2016, or Office 365), and Business Intelligence (BI) systems, such as Tableau and Microsoft Power BI.

Click below to see query examples using specific tools:

- *Browser query examples* (see "*Retrieve data via browser query*" on page 232)
- *Postman query examples* (see "*Retrieve data using Postman*" on page 233)
- *Excel query examples* (see "*Retrieve data using Excel*" on page 234)
- *PowerBI query example* (see "*Retrieve data using PowerBI*" on page 237)

Retrieve data via browser query

You can simply type an OData HTTP request in the address bar of a modern browser to query Historian Data API.

Note: Retrieval endpoints are unique for each Insight solution. The following examples use the endpoint "https://online.wonderware.com/s/ik97r5", but yours will be different. Learn how to find the basic-authentication retrieval endpoint for your solution.

- **List root entities**

  This example lists the entities you can explore and submit additional queries against.

  ```
  https://online.wonderware.com/s/ik97r5/apis/historian/v2
  ```

  For more information, see *Resources* (see "*Retrieval resources*" on page 189).

- **List data sources (tag groups)**

  This shows the data sources (labeled "TagGroups" in this API) for your Insight solution. If you have accounts for more than one solution, this shows the data sources for your original Insight solution.

  ```
  https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagGroups
  ```

- **List tag metadata for all tags**

  This shows metadata for all tags in all data sources for your original Insight solution.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/Tags
```

- **List tag metadata for a data source**

  This shows metadata for tags in a particular data source. The data source in this example is named Baytown.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/Tags?$filter=Source+eq+'Baytown'
```

- **List tag values for a data source**

  This example expands on the last one. It uses a "$filter" clause that specifies a particular tag named tank_level within the Baytown data source. The result is a list of values for the tank_level tag.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/ProcessValues?$filter=FQN+eq+'Baytown.tank_level'
```

- **List tag values with analog summary data for a data source**

  This example result is a list of values with analog summary for the tank_level tag.

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/AnalogSummary?$filter=FQN+eq+'Baytown.tank_level'
```

  Notice that this example uses a fully qualified name ("Baytown.tank_level"), which is a combination of a data source name ("Baytown") and a tagname ("tank_level").

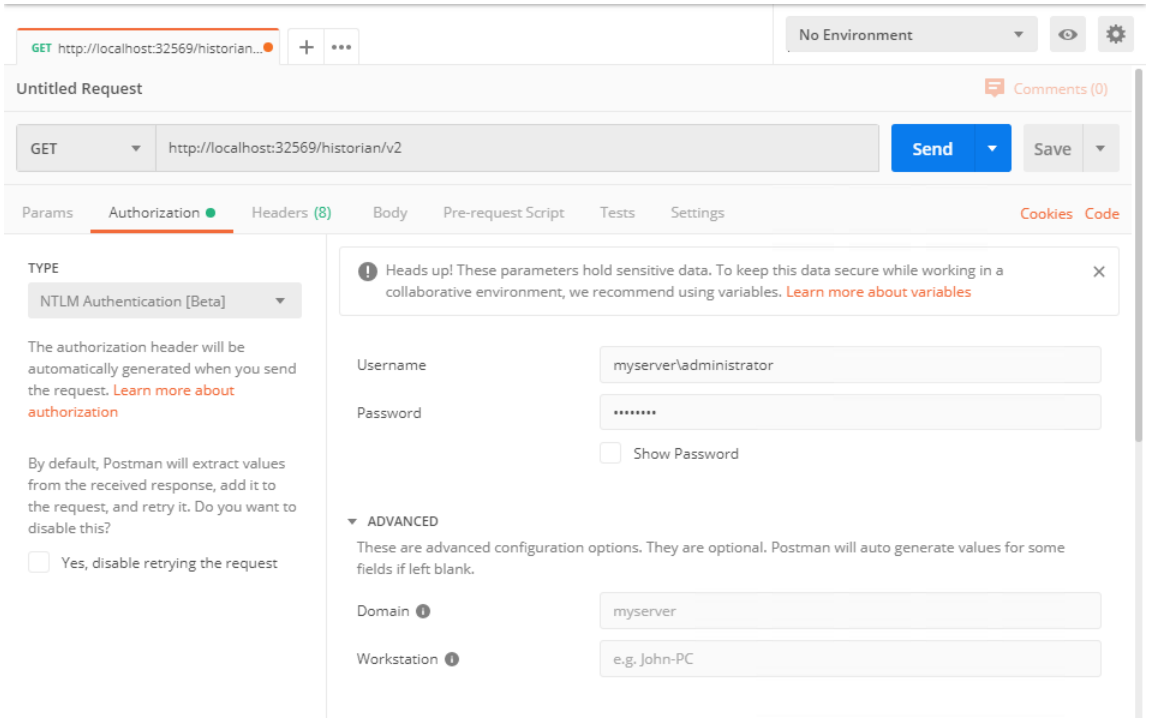
Retrieve data using Postman

A great way to explore and learn about the AVEVA Insight APIs is by using a simple client, such as Postman. Postman is a free developer tool allows you to query any API. You can download Postman from *https://postman.com/ https://www.getpostman.com/*.

To query the AVEVA Insight APIs using Postman

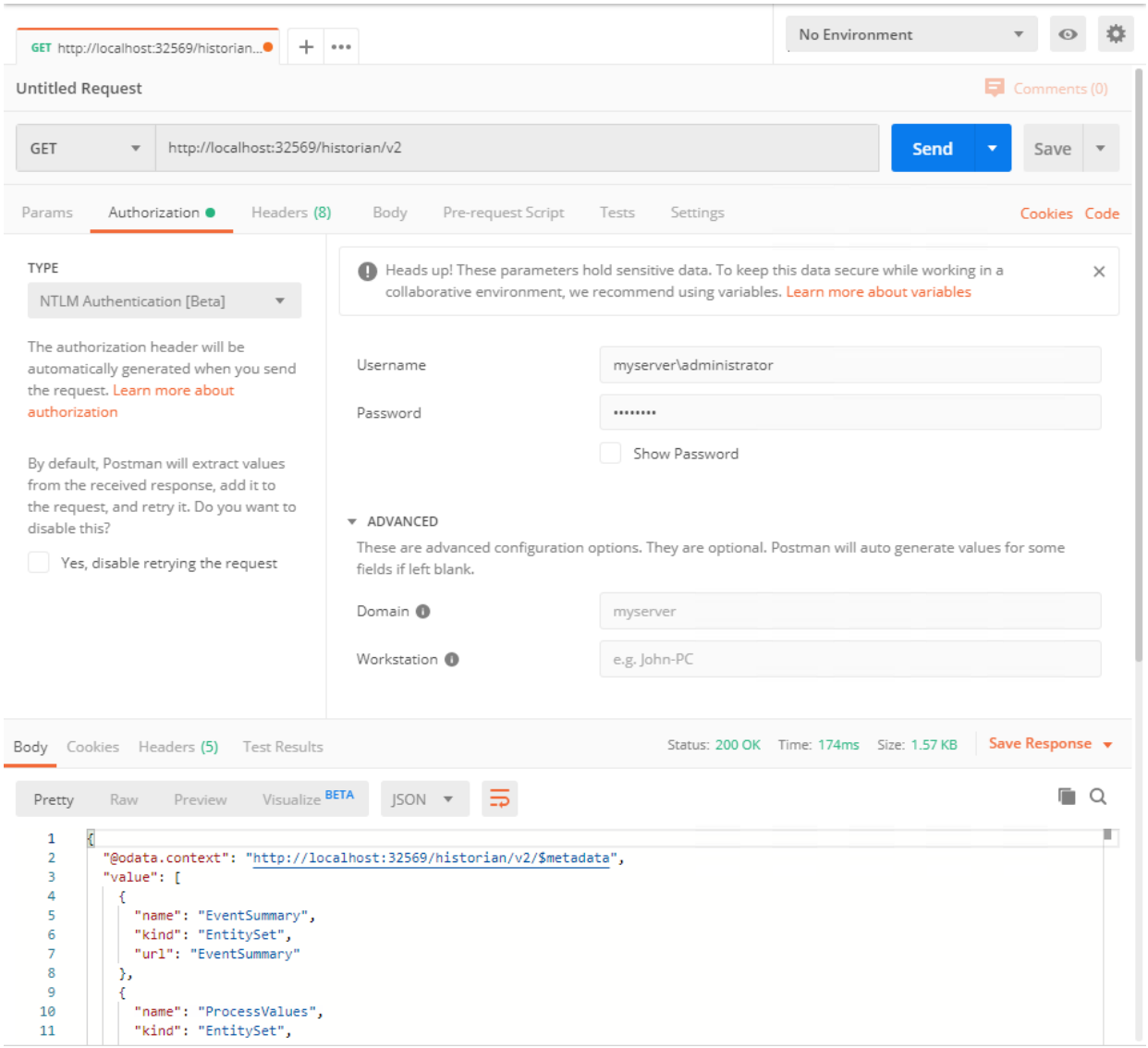
1. Next to the GET button, type this address:

```
http://localhost:32569/historian/v2
```



2. From the **Type** dropdown list, select **NTLM Authentication [Beta]** to specify the authentication to use.
3. Enter the username and password for a user account with sufficient privileges to access the API.
4. In the upper-right of the screen, select **Send**.

   The result is a catalog of data entities from AVEVA Insight.



Retrieve data using Excel

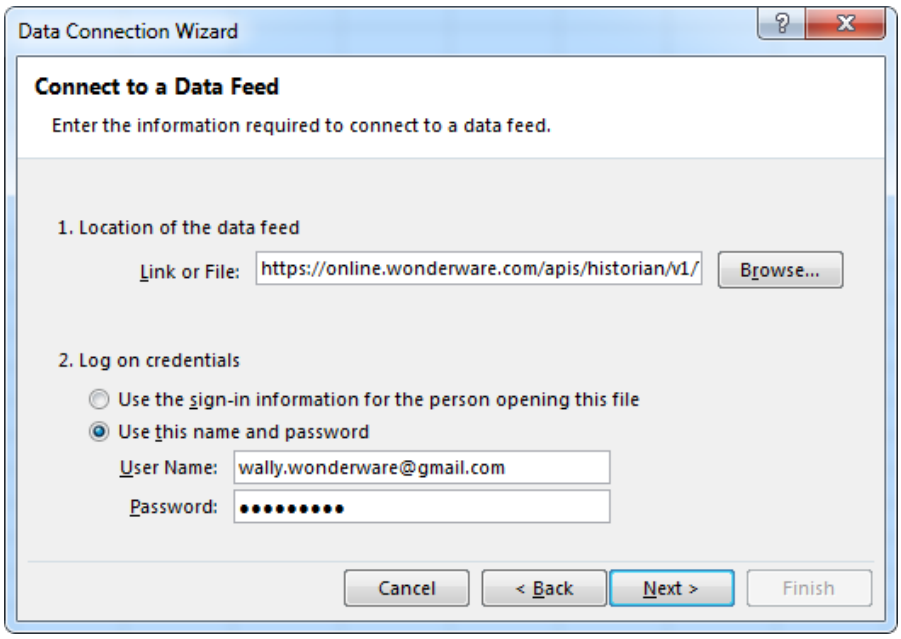
To manually import Insight data to Excel

1. Open Excel 2013 or later.
2. From **Data**, choose **Get External Data**, choose **From Other Sources**, and then choose **From OData Data Feed**.
3. In the **Link or File** box, specify a URL for your query, formatted as an atom feed (that is, end with "$format=atom'").

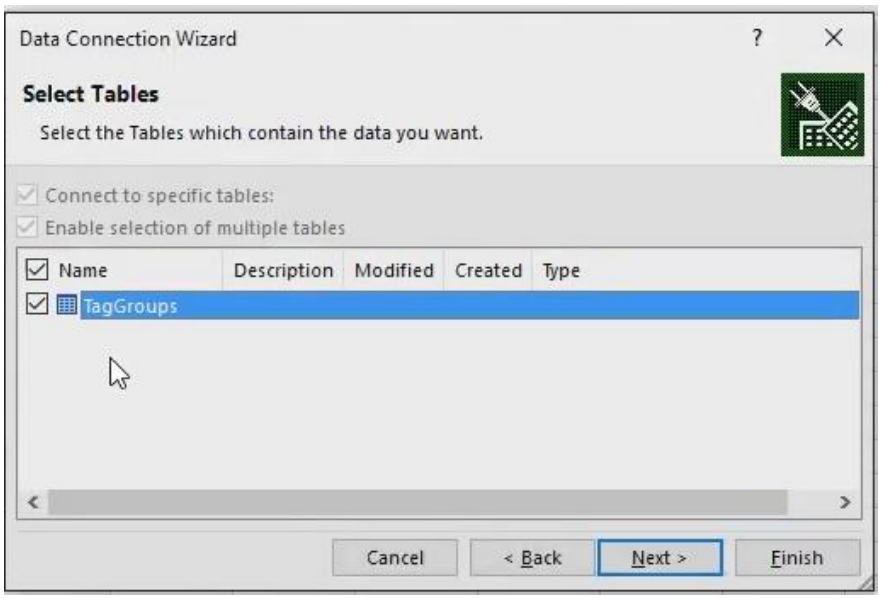
   Here is a query example:

```
https://online.wonderware.com/s/ik97r5/apis/historian/v2/TagGroups?$format=atom
```

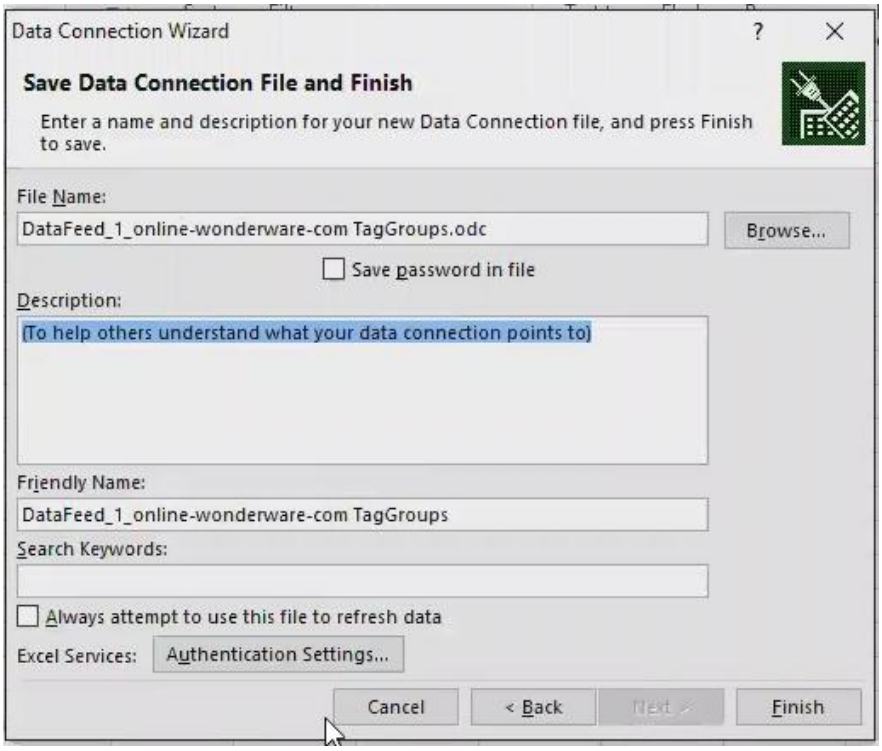
Note: Retrieval endpoints are unique for each Insight solution. This example uses the endpoint "https://online.wonderware.com/s/ik97r5", but yours will be different. Learn how to find the basic-authentication retrieval endpoint for your solution.



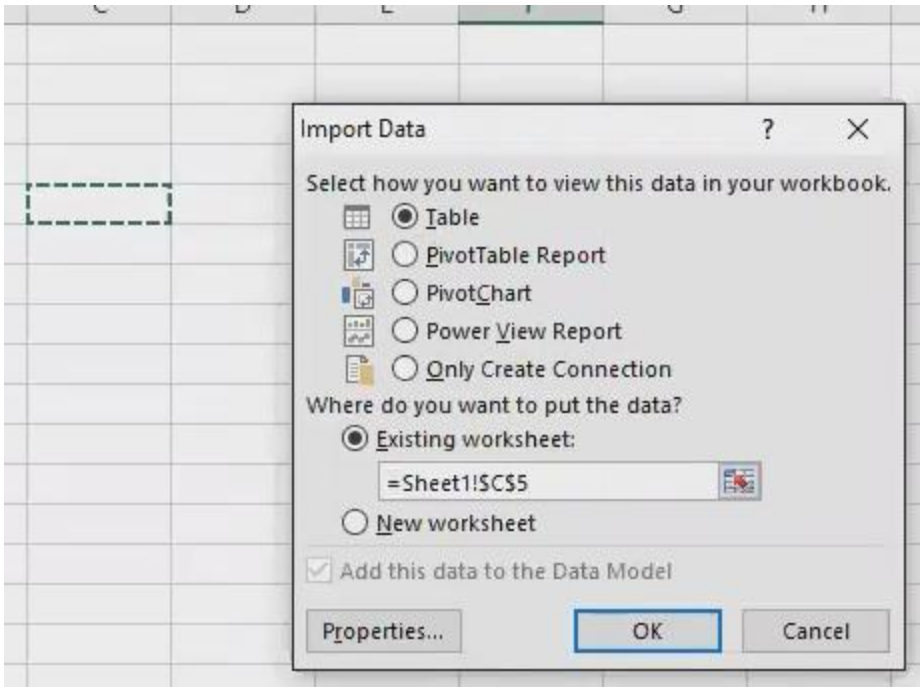
4. Click **Use this name and password**, and specify your Insight user name and password. Click **Next**.
5. In the **Select Tables** box, mark the element to include. Click **Next**.



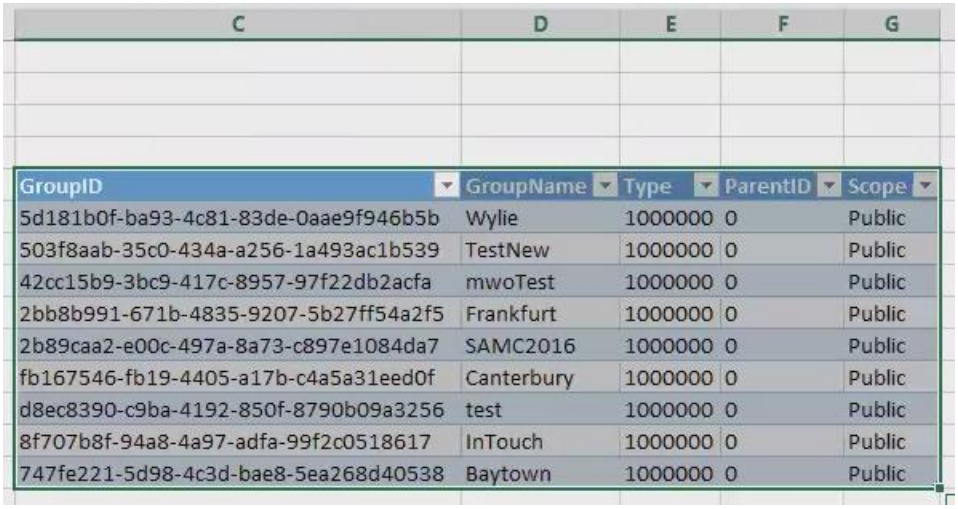
6. Review the information displayed on **Save Data Connection File and Finish** and then click **Finish**. Excel creates an ODC (OData Connection) file.



7. Specify how and where to place your data in the spreadsheet. Click **OK**.



Excel will place the data where you specified:



GroupID	GroupName	Type	ParentID	Scope
5d181b0f-ba93-4c81-83de-0aae9f946b5b	Wylie	1000000	0	Public
503f8aab-35c0-434a-a256-1a493ac1b539	TestNew	1000000	0	Public
42cc15b9-3bc9-417c-8957-97f22db2acfa	mwoTest	1000000	0	Public
2bb8b991-671b-4835-9207-5b27ff54a2f5	Frankfurt	1000000	0	Public
2b89caa2-e00c-497a-8a73-c897e1084da7	SAMC2016	1000000	0	Public
fb167546-fb19-4405-a17b-c4a5a31eed0f	Canterbury	1000000	0	Public
d8ec8390-c9ba-4192-850f-8790b09a3256	test	1000000	0	Public
8f707b8f-94a8-4a97-adfa-99f2c0518617	InTouch	1000000	0	Public
747fe221-5d98-4c3d-bae8-5ea268d40538	Baytown	1000000	0	Public

Retrieve data using PowerBI

You can use PowerBI to retrieve Insight data. For syntax examples, see the Query Examples section below.

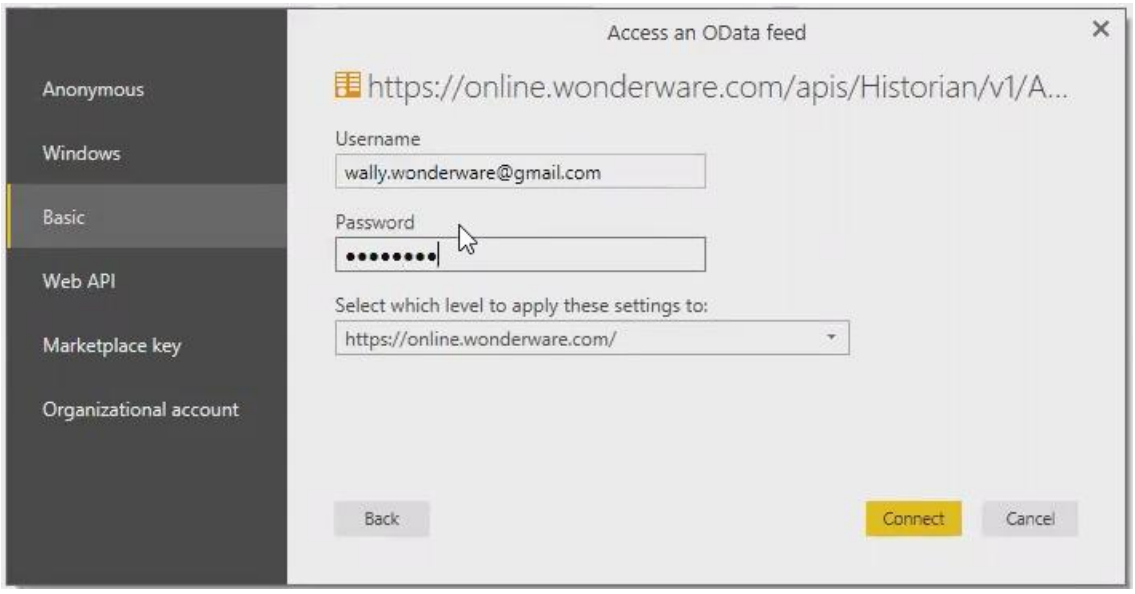
Note: You will need a query endpoint from AVEVA Insight for your query.

To import Insight data to Power BI

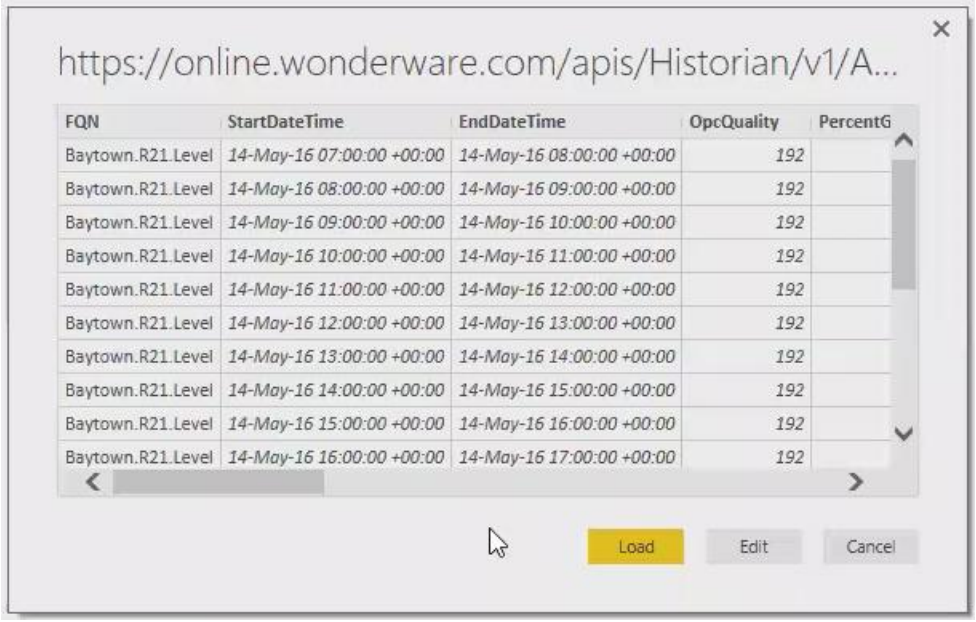
1. In AVEVA Insight, identify the retrieval endpoint for your Insight solution. You will use this in step 4.
2. Open PowerBI.
3. Click **Get Data**, and choose **OData Feed**.
4. In **URL**, specify the retrieval endpoint (from step 1) followed by any query parameters. Click **OK**.

   See the "Query examples" section below.

5. Specify **Basic Authentication**, and provide your Insight username and password.

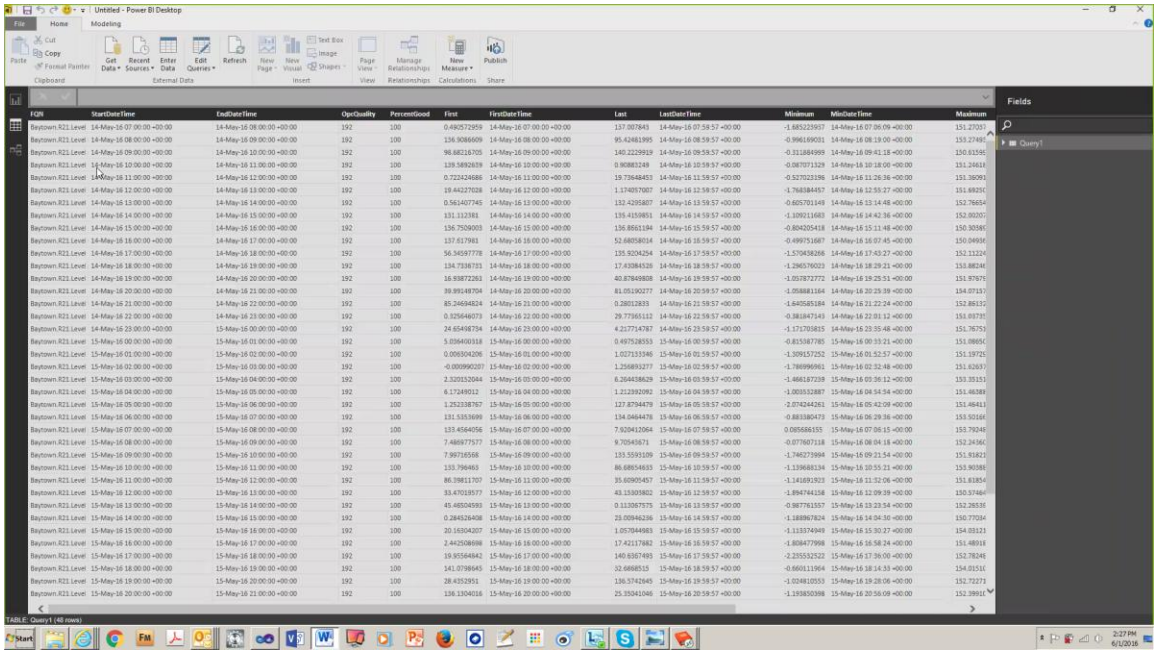


6. Click **Connect**.
7. Power BI shows a preview of your data retrieved from Insight.



8. Click **Load**.

9. In the upper-left corner, click  to view the table of results.



Query examples

- **Using GET method with ProcessValues:**

```
Historian/v2/ProcessValues?TagFilter=EngUnit eq 'rpm'&DurationHours=2

Historian/v2/ProcessValues?$filter=DateTime le
2019-01-15T03:57:29Z&TagFilter=startswith(Description,
'Pump')&DurationHours=10

Historian/v2/ProcessValues?$filter=DateTime ge 2019-01-15T03:57:29Z and
DateTime le 2019-01-17T03:50:54.881Z&TagFilter=startswith(Location,
'/Houston/Mixing')

Historian/v2/ProcessValues?$filter=DateTime ge
2019-01-15T03:57:29Z&TagFilter=startswith(FQN, 'testds1')&DurationHours=5

Historian/v2/ProcessValues?TagFilter=endswith('Level',FQN)
```

- **Using GET method with AnalogSummary:**

```
Historian/v2/AnalogSummary?TagFilter=EngUnit eq 'rpm'&DurationHours =2

Historian/v2/AnalogSummary?$filter=EndDateTime le
2019-01-15T03:57:29Z&TagFilter= startswith(Description,
'Pump')&DurationHours=10

Historian/v2/AnalogSummary?$filter=StartDateTime ge 2019-01-15T03:57:29Z
and EndDateTime le 2019-01-17T03:50:54.881Z&TagFilter=startswith(Location,
"/Houston/Mixing")

Historian/v2/AnalogSummary?$filter= StartDateTime ge
2019-01-15T03:57:29Z&TagFilter=startswith(FQN,
'testds1')&DurationInHour=5
```

```
Historian/v2/AnalogSummary?TagFilter=endswith(FQN, 'Level')
```

- **Using GET method with StateSummary:**

```
Historian/v2/StateSummary?TagFilter=EngUnit eq 'rpm'& DurationHours =2

Historian/v2/StateSummary?$filter=EndDateTime le
2019-01-15T03:57:29Z&TagFilter=startswith(Description, 'Pump')&
DurationHours =10

Historian/v2/StateSummary?$filter=StartDateTime ge 2019-01-15T03:57:29Z
and EndDateTime le 2019-01-17T03:50:54.881Z&TagFilter=startswith(Location,
'/Houston/Mixing')

Historian/v2/StateSummary?$filter= StartDateTime ge
2019-01-15T03:57:29Z&TagFilter=startswith(FQN, 'testds1')&DurationHours=5

Historian/v2/StateSummary?TagFilter=endswith(FQN, 'Level')
```

Notes about post-retrieval filtering

Any post-retrieval filtering may require you to also change the data type for a specific the column in the results list.

For example, suppose you ran this query:

```
/Historian/v2/ProcessValues?$filter=OPCQuality eq
192&DurationHours=24&Resolution=3600000&TagFilter=
endswith(TagName,'%23testtag')
```

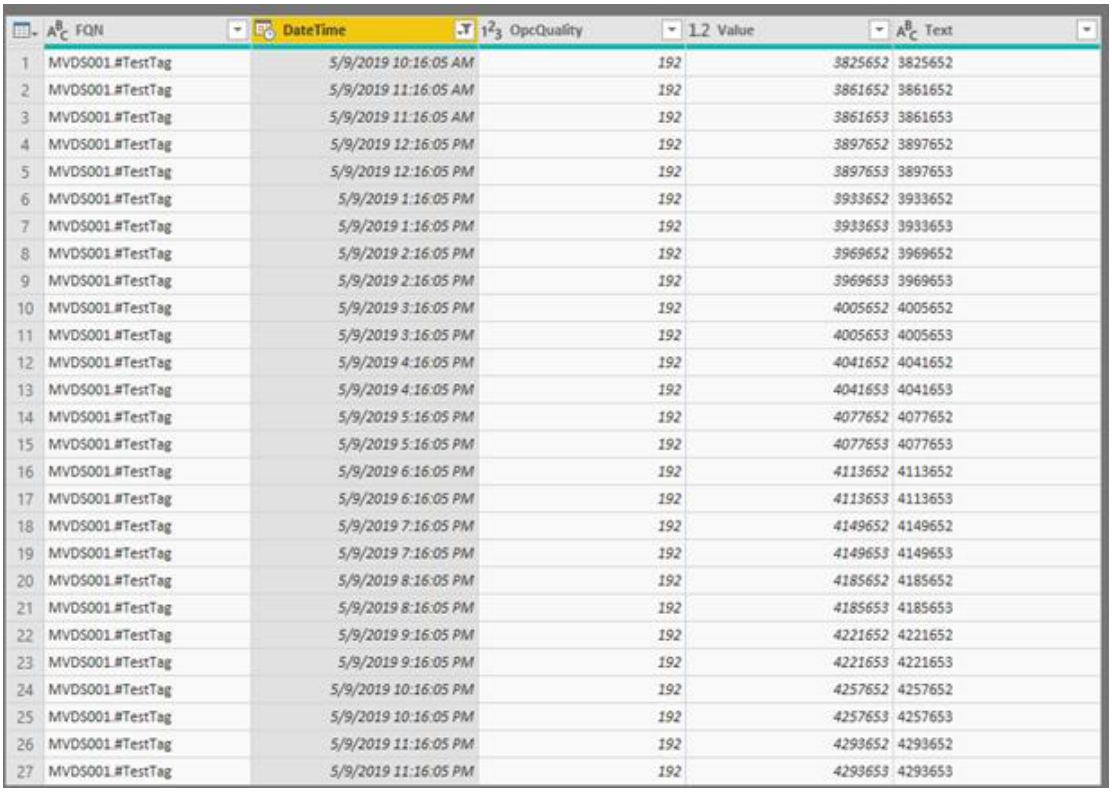
The results would be:

	FQN	DateTime	OpcQuality	Value	Text
1	MVDS001.#TestTag	5/9/2019 5:16:05 PM +00:00	192	3825652	3825652
2	MVDS001.#TestTag	5/9/2019 6:16:05 PM +00:00	192	3861652	3861652
3	MVDS001.#TestTag	5/9/2019 6:16:05 PM +00:00	192	3861653	3861653
4	MVDS001.#TestTag	5/9/2019 7:16:05 PM +00:00	192	3897652	3897652
5	MVDS001.#TestTag	5/9/2019 7:16:05 PM +00:00	192	3897653	3897653
6	MVDS001.#TestTag	5/9/2019 8:16:05 PM +00:00	192	3933652	3933652
7	MVDS001.#TestTag	5/9/2019 8:16:05 PM +00:00	192	3933653	3933653
8	MVDS001.#TestTag	5/9/2019 9:16:05 PM +00:00	192	3969652	3969652
9	MVDS001.#TestTag	5/9/2019 9:16:05 PM +00:00	192	3969653	3969653
10	MVDS001.#TestTag	5/9/2019 10:16:05 PM +00:00	192	4005652	4005652
11	MVDS001.#TestTag	5/9/2019 10:16:05 PM +00:00	192	4005653	4005653
12	MVDS001.#TestTag	5/9/2019 11:16:05 PM +00:00	192	4041652	4041652
13	MVDS001.#TestTag	5/9/2019 11:16:05 PM +00:00	192	4041653	4041653
14	MVDS001.#TestTag	5/10/2019 12:16:05 AM +00:00	192	4077652	4077652
15	MVDS001.#TestTag	5/10/2019 12:16:05 AM +00:00	192	4077653	4077653
16	MVDS001.#TestTag	5/10/2019 1:16:05 AM +00:00	192	4113652	4113652
17	MVDS001.#TestTag	5/10/2019 1:16:05 AM +00:00	192	4113653	4113653
18	MVDS001.#TestTag	5/10/2019 2:16:05 AM +00:00	192	4149652	4149652

Then suppose you wanted to filter the "DateTime" column to include only yesterday's date. You must change the data type of the DateTime column first, and then select the date you want:

1. Right-click the "DateTime" column header, select **Change Type**. and then click **Date/Time**.
2. Right-click the "DateTime" column and choose **Yesterday** from the displayed list.

Then the filtering will take effect.



	FQN	DateTime	OpcQuality	Value	Text
1	MVDS001.#TestTag	5/9/2019 10:16:05 AM	192	3825652	3825652
2	MVDS001.#TestTag	5/9/2019 11:16:05 AM	192	3861652	3861652
3	MVDS001.#TestTag	5/9/2019 11:16:05 AM	192	3861653	3861653
4	MVDS001.#TestTag	5/9/2019 12:16:05 PM	192	3897652	3897652
5	MVDS001.#TestTag	5/9/2019 12:16:05 PM	192	3897653	3897653
6	MVDS001.#TestTag	5/9/2019 1:16:05 PM	192	3933652	3933652
7	MVDS001.#TestTag	5/9/2019 1:16:05 PM	192	3933653	3933653
8	MVDS001.#TestTag	5/9/2019 2:16:05 PM	192	3969652	3969652
9	MVDS001.#TestTag	5/9/2019 2:16:05 PM	192	3969653	3969653
10	MVDS001.#TestTag	5/9/2019 3:16:05 PM	192	4005652	4005652
11	MVDS001.#TestTag	5/9/2019 3:16:05 PM	192	4005653	4005653
12	MVDS001.#TestTag	5/9/2019 4:16:05 PM	192	4041652	4041652
13	MVDS001.#TestTag	5/9/2019 4:16:05 PM	192	4041653	4041653
14	MVDS001.#TestTag	5/9/2019 5:16:05 PM	192	4077652	4077652
15	MVDS001.#TestTag	5/9/2019 5:16:05 PM	192	4077653	4077653
16	MVDS001.#TestTag	5/9/2019 6:16:05 PM	192	4113652	4113652
17	MVDS001.#TestTag	5/9/2019 6:16:05 PM	192	4113653	4113653
18	MVDS001.#TestTag	5/9/2019 7:16:05 PM	192	4149652	4149652
19	MVDS001.#TestTag	5/9/2019 7:16:05 PM	192	4149653	4149653
20	MVDS001.#TestTag	5/9/2019 8:16:05 PM	192	4185652	4185652
21	MVDS001.#TestTag	5/9/2019 8:16:05 PM	192	4185653	4185653
22	MVDS001.#TestTag	5/9/2019 9:16:05 PM	192	4221652	4221652
23	MVDS001.#TestTag	5/9/2019 9:16:05 PM	192	4221653	4221653
24	MVDS001.#TestTag	5/9/2019 10:16:05 PM	192	4257652	4257652
25	MVDS001.#TestTag	5/9/2019 10:16:05 PM	192	4257653	4257653
26	MVDS001.#TestTag	5/9/2019 11:16:05 PM	192	4293652	4293652
27	MVDS001.#TestTag	5/9/2019 11:16:05 PM	192	4293653	4293653

Querying History Blocks via SQL Server Reporting Services Extension

Historian includes an extension for SQL Server Reporting Services to enable SQL queries against the OData interface used for data stored in history blocks.

To configure the extension

1. Open the Configurator.
2. Under Historian, click the **Reporting** node.
3. Select the versions of Visual Studio installed with your SQL Server Reporting Services.
4. Click **Configure**.

To use the extension

1. Define a new Reporting Services data source.
2. Select "Historian OData Provider" as the source type.
3. Click **Historian** and then click **Extension**.
4. Enter a connection string in the form "Server=host:port"; for example:

```
Server=localhost:32569
```

Three Ways To Query the Source

You can query the source using one of these options:

- **Use a SQL-style query**. For example:

```
select * from Events where Source_Area='Site2' and EventTime >=
'2014-08-10T05:56:21.302Z'
```

- **Use an OData query**. This OData query example is equivalent to the query above:

```
http://Server1:32569/Historian/v1/Events?$filter=Source_Area+eq+%27Sit
e2%27+and EventTime+ge+datetimeoffset%272014-08-10T05:56:21.302Z%27
```

Note: For queries from a web browser, use "%20" to indicate a space. Use "%27" to indicate a single quote.
If you are using the JSONView viewer in the Chrome browser, you can use a plus sign (+) to indicate a space to make the URI string more readable.

- **Use an SQL syntax used within the URL**. For example, this is equivalent to the other two queries above:

```
http://Server1:32569/Historian/v1/Events?sql=select+*+from
Events+where
Source_Area=%27ite2%27+and+EventTime+>=+%272014-08-10T05:56:21.302Z%27
```

Retrieval errors

The OData error codes listed in the following table may be returned by an operation on any of the storage services.

Error code	HTTP status code	User message
ConditionNotMet	Not Modified (304)	The condition specified in the conditional header(s) was not met for a read operation.
MissingRequiredQueryParameter	Bad Request (400)	A required query parameter was not specified for this request.
UnsupportedQueryParameter	Bad Request (400)	One of the query parameters specified in the request URI is not supported.
InvalidQueryParameterValue	Bad Request (400)	An invalid value was specified for one of the query parameters in the request URI.
OutOfRangeQueryParameterValue	Bad Request (400)	A query parameter specified in the request URI is outside the permissible range.
RequestUrlFailedToParse	Bad Request (400)	The url in the request could not be parsed.
InvalidUri	Bad Request (400)	The requested URI does not represent any resource on the server.
InvalidHttpVerb	Bad Request (400)	The HTTP verb specified was not recognized by the server.

Error code	HTTP status code	User message
OutOfRangeInput	Bad Request (400)	One of the request inputs is out of range.
InvalidAuthenticationInfo	Bad Request (400)	The authentication information was not provided in the correct format. Verify the value of Authorizationheader.
InvalidInput	Bad Request (400)	One of the request inputs is not valid.
Unauthorized	Unauthorized (401)	Unauthorized: Access is denied due to invalid credentials.
ResourceNotFound	Not Found (404)	The specified resource does not exist.
InternalError	Internal Server Error (500)	The server encountered an internal error. Please retry the request.
OperationTimedOut	Internal Server Error (500)	The operation could not be completed within the permitted time.
ServerBusy	Service Unavailable (503)	The server is currently unable to receive requests. Please retry your request.