Wonderware Historian Concepts Guide

Invensys Systems, Inc.

Revision F

Last Revision: December 3, 2009

Copyright

© 2002-2005, 2009 Invensys Systems, Inc. All Rights Reserved.

All rights reserved. No part of this documentation shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Invensys Systems, Inc. No copyright or patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this documentation, the publisher and the author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

The information in this documentation is subject to change without notice and does not represent a commitment on the part of Invensys Systems, Inc. The software described in this documentation is furnished under a license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of these agreements.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200

http://www.wonderware.com

For comments or suggestions about the product documentation, send an e-mail message to productdocs@wonderware.com.

Trademarks

All terms mentioned in this documentation that are known to be trademarks or service marks have been appropriately capitalized. Invensys Systems, Inc. cannot attest to the accuracy of this information. Use of a term in this documentation should not be regarded as affecting the validity of any trademark or service mark.

Alarm Logger, ActiveFactory, ArchestrA, Avantis, DBDump, DBLoad, DT Analyst, Factelligence, FactoryFocus, FactoryOffice, FactorySuite, FactorySuite A^2, InBatch, InControl, IndustrialRAD, IndustrialSQL Server, InTouch, MaintenanceSuite, MuniSuite, QI Analyst, SCADAlarm, SCADASuite, SuiteLink, SuiteVoyager, WindowMaker, WindowViewer, Wonderware, Wonderware Factelligence, and Wonderware Logger are trademarks of Invensys plc, its subsidiaries and affiliates. All other brands may be trademarks of their respective owners.

Contents

Welcome 11

Wonderware Historian Documentation Set 11

Documentation Conventions 12

Technical Support 13

Chapter 1 Introduction 15

The Wonderware Historian Solution 15

Process Data 16

About Relational Databases 16

Limitations of Relational Databases 17

Wonderware Historian as a Real-Time Relational Database 17

Integration with Microsoft SQL Server 18

Support for SQL Clients 19

Wonderware Historian Subsystems 19

Chapter 2 System-Level Functionality 21

About Tags 21

Types of Tags 21

Sources of Tag Values 22

Naming Conventions for Tags 23

Security 24

Windows Operating System Security 24

SQL Server Security 25

Management Console Security 31
Default Access Rights for Different Operating Systems 32
Time Handling...32
System Parameters 33
System Messages...38
Wonderware Historian Services 40
The System Driver and System Tags..............................42
Error Count Tags 42
Date Tags...42
Time Tags 43
Storage Space Tags 43
I/O Statistics Tags..43
System Monitoring Tags...44
Miscellaneous (Other) Tags 45
Event Subsystem Tags..46
Replication Subsystem Tags.......................................47
Performance Monitoring Tags 48
Supported Protocols 50
Modification Tracking..51
Modification Tracking for Configuration Changes.......52
Modification Tracking for Historical Data Changes 53
Data Quality 54
Viewing Quality Values and Details............................55
Acquisition and Storage of Quality Information 59
Client-Side Quality...61

Chapter 3 Configuration Subsystem 63

Configuration Subsystem Components...........................64
About the Runtime and Holding Databases 65
The Runtime Database 65
The Holding Database...66
About the Configuration Service 67
Dynamic Configuration..67
Effects of Configuration Changes on the System 68
Cases in Which Configuration Changes are not Committed..70

Chapter 4 Data Acquisition Subsystem 71
Data Acquisition Components 72
Data Acquisition from I/O Servers 73
I/O Server Addressing 74
About IDASs 75
I/O Server Redundancy 86
Redirecting I/O Servers to InTouch HMI Software 87
Time Synchronization for Data Acquisition.................87
Data Acquisition by Means of INSERT and UPDATE Statements..90
Data Acquisition from MDAS 90
Importing Data from a CSV File 91

Chapter 5 Data Storage Subsystem....................... 93
Storage Subsystem Components 94
Storage Data Categories 94
About the Real-Time Data Window..............................97
Data Modification and Versioning..................................98
Storage Modes 99
"Forced" Storage 99
Delta Storage 100
Time and Value Deadbands for Delta Storage 100
"Swinging Door" Deadband for Delta Storage 101
Cyclic Storage 111
Data Conversions and Reserved Values for Storage 112
History Blocks 113
History Block Notation...113
History Block Creation...114
History Block Storage Locations 115
Automatic Deletion of History Blocks 117
About the Active Image...119
Automatic Resizing of the Active Image 120
How the Active Image Storage Option Affects Data Retrieval 121
Dynamic Configuration Effects on Storage...................122
Memory Management for Data Storage........................122
About Snapshot Files 124
How Snapshot Files are Updated 126

Chapter 6 Data Retrieval Subsystem 127

Data Retrieval Components.. 128
Data Retrieval Features 129
History Blocks: A SQL Server Remote Data Source 129
Retrieval Service 130
About the Wonderware Historian OLE DB Provider.... 130
  Extension (Remote) Tables for History Data 132
  Query Syntax for the Wonderware Historian OLE DB Provider 133
  Wonderware Historian OLE DB Provider Unsupported Syntax and Limitations 137
  Linking the Wonderware Historian OLE DB Provider to the Microsoft SQL Server 146
Wonderware Historian Time Domain Extensions......... 147
Wonderware Historian I/O Server 149

Chapter 7 Data Retrieval Options 151

Understanding Retrieval Modes.................................... 151
  Cyclic Retrieval 152
  Delta Retrieval 156
  Full Retrieval... 163
  Interpolated Retrieval.. 165
  “Best Fit” Retrieval 171
  Average Retrieval.. 176
  Minimum Retrieval 182
  Maximum Retrieval 188
  Integral Retrieval... 194
  Slope Retrieval 197
  Counter Retrieval... 200
  ValueState Retrieval.. 205
  RoundTrip Retrieval 212
Understanding Retrieval Options 217
  Which Options Apply to Which Retrieval Modes?...... 217
  Using Retrieval Options in a Transact-SQL Statement 218
  Cycle Count (X Values over Equal Time Intervals) (wwCycleCount) 219
  Resolution (Values Spaced Every X ms) (wwResolution)... 222
  About “Phantom” Cycles 224
  Time Deadband (wwTimeDeadband) 227
  Value Deadband (wwValueDeadband)........................ 231

History Version (wwVersion) 235
Interpolation Type (wwInterpolationType)................237
Timestamp Rule (wwTimestampRule).......................240
Time Zone (wwTimeZone) 242
Quality Rule (wwQualityRule) 244
State Calculation (wwStateCalc)................................252
Analog Value Filtering (wwFilter) 254
Selecting Values for Analog Summary Tags (wwValueSelector) 261
Edge Detection for Events (wwEdgeDetection) 264

Chapter 8 Query Examples 275

Querying the History Table 275
Querying the Live Table 276
Querying the WideHistory Table...................................277
Querying Wide Tables in Delta Retrieval Mode 278
Querying the AnalogSummaryHistory View 279
Querying the StateSummaryHistory View 280
Using an Unconventional Tagname in a Wide Table Query 281
Using an INNER REMOTE JOIN................................281
Setting Both a Time and Value Deadband for Retrieval 282
Using wwResolution, wwCycleCount, and wwRetrievalMode in the Same Query 285
Determining Cycle Boundaries.....................................286
Mixing Tag Types in the Same Query...........................286
Using a Criteria Condition on a Column of Variant Data...287
Using DateTime Functions 288
Using the GROUP BY Clause.......................................290
Using the COUNT() Function.......................................290
Using an Arithmetic Function 291
Using an Aggregate Function 292
Making and Querying Annotations 294
Using Comparison Operators with Delta Retrieval 294
Using Comparison Operators with Cyclic Retrieval and Cycle Count 299
Using Comparison Operators with Cyclic Retrieval and Resolution...302
SELECT INTO from a History Table 306

Moving Data from a SQL Server Table to an Extension Table 307
Using Server-Side Cursors 308
Using Stored Procedures in OLE DB Queries 310
Querying Data to a Millisecond Resolution using SQL Server 2005 310
Getting Data from the OPCQualityMap Table 312
Using Variables with the Wide Table 312
Retrieving Data Across a Data "Hole" 313
Returned Values for Non-Valid Start Times 315
Retrieving Data from History Blocks and the Active Image 315
Querying Aggregate Data in Different Ways 316

Chapter 9 Replication Subsystem 319
About Tiered Historians 319
How Tags are Used During Replication 321
Simple Replication 323
Summary Replication 324
Analog Summary Replication 325
State Summary Replication 326
Replication Schedules 327
Replication Schedules and Daylight Savings Time 328
Replication Groups 330
How Replication is Handled for Different Types of Data 331
Streaming Replication 332
Queued Replication 332
Tag Configuration Synchronization between Tiered Historians 333
Replication Components 334
Replication Run-time Operations 335
Replication Latency 336
Replication Delay for “Old” Data 336
Continuous Operation 336
Overflow Protection 337
Security for Data Replication 337
Using Summary Replication instead of Event-Based Summaries 338

Chapter 10 Event Subsystem 339

Event Subsystem Components 340
Uses for the Event Subsystem 341
Event Subsystem Features and Benefits 342
Event Subsystem Performance Factors 343
Event Tags 344
Event Detectors 345
SQL-Based Detectors 345
Schedule Detectors 348
External Detectors 349
Event Actions...349
Generic SQL Actions 349
Snapshot Actions 350
E-mail Actions 350
Deadband Actions..351
Summary Actions 351
Event Action Priorities...353
Event Subsystem Resource Management......................353
Detector Thread Pooling 354
Action Thread Pooling...355
Event Subsystem Database Connections....................356
Handling of Event Overloads and Failed Queries......356
Event Subsystem Variables..358

Index 361

Welcome

This guide provides information about the general architecture of the Wonderware Historian and describes the different subsystems and components that make up the product. This guide can be used as a reference guide for all conceptual information about the Wonderware Historian components.

Wonderware Historian Documentation Set

The Wonderware Historian documentation set includes the following guides:

- *Wonderware Historian Installation Guide* (InSQLInstall.pdf). This guide provides information on installing the Wonderware Historian, including hardware and software requirements and migration instructions.
- *Wonderware Historian Concepts Guide* (InSQLConcepts.pdf). This guide provides an overview of the entire Wonderware Historian system and describes each of the subsystems in detail.
- *Wonderware Historian Administration Guide* (InSQLAdmin.pdf). This guide describes how to administer and maintain an installed Wonderware Historian, such as configuring data acquisition and storage, managing security, and monitoring the system.
- *Wonderware Historian Database Reference* (InSQLDatabase.pdf). This guide provides documentation for all of the Wonderware Historian database entities, such as tables, views, and stored procedures.

- *Wonderware Historian Glossary* (InSQLGlossary.pdf). This guide provides definitions for terms used throughout the documentation set.

In addition, the *Wonderware® ArchestrA License Manager Guide* (License.pdf) describes the ArchestrA License Manager and how to use it to install, maintain, and delete licenses and license servers on local and remote computers.

A PDF file for each of these guides is available on the Wonderware Historian installation CD. You can easily print information from the PDF files. The Wonderware Historian documentation is also provided as an online help file, which can be accessed from the System Management Console management tool.

Documentation Conventions

This documentation uses the following conventions:

Convention	Used for
Initial Capitals	Paths and file names.
Bold	Menus, commands, dialog box names, and dialog box options.
`Monospace`	Code samples and display text.

Technical Support

Wonderware Technical Support offers a variety of support options to answer any questions on Wonderware products and their implementation.

Before you contact Technical Support, refer to the relevant section(s) in this documentation for a possible solution to the problem. If you need to contact technical support for help, have the following information ready:

- The type and version of the operating system you are using.
- Details of how to recreate the problem.
- The exact wording of the error messages you saw.
- Any relevant output listing from the Log Viewer or any other diagnostic applications.
- Details of what you did to try to solve the problem(s) and your results.
- If known, the Wonderware Technical Support case number assigned to your problem, if this is an ongoing problem.

Chapter 1

Introduction

The Wonderware Historian, formally known as IndustrialSQL Server™, bridges the gap between a real-time high-volume plant monitoring environment and an open, flexible business information environment. The historian:

- Acquires plant data from high-speed Wonderware I/O Servers, DAServers, InTouch HMI software, Wonderware Application Server, and other devices.
- Compresses and stores data.
- Responds to SQL requests for plant data.

The historian also contains event, summary, configuration, security, backup, and system monitoring information.

The historian is tightly coupled to Microsoft SQL Server.

The Wonderware Historian Solution

The Wonderware Historian is a real-time relational database that stores plant data. The historian acquires and stores process data at full resolution or at a specified resolution and provides real-time and historical plant data together with configuration, event, summary, and associated production data to client applications on the desktop. The historian combines the power and flexibility of Microsoft SQL Server with the high speed acquisition and efficient data compression characteristics of a real-time system.

Process Data

Process data is any relevent information to successfully run a process. The following information is considered to be process data:

- Real-time data - What is the current value of this tag?
- Historical data - What was the value of this tag every second last Monday?
- Summary data - What is the average of each of these five tags?
- Event data - When did that boiler trip?
- Configuration data - How many I/O Servers am I using and what are their types?

To improve performance and quality while reducing cost, process data must be available for analysis. Process data is typically analyzed to determine:

- Process analysis, diagnostics, and optimization.
- Predictive and preventive equipment maintenance.
- Product and process quality (SPC/SQC).
- Health and safety; environmental impact (EPA/FDA).
- Production reporting.
- Failure analysis.

About Relational Databases

A relational database management system (RDBMS) stores data in multiple tables that are related or linked together. Storing and accessing information in multiple tables makes data storage and maintenance more efficient than if all of the information was stored in a single large table. For example, Microsoft SQL Server is a relational database.

SQL, is the language to communicate with relational databases. SQL is an industry "super-standard," supported by hundreds of software vendors. SQL provides an openness unmatched in the plant environment. Relational databases are mature and are the accepted IT workhorses in database applications today. Power and flexibility are far superior in SQL than in the proprietary interfaces that have come out of the plant environment.

Limitations of Relational Databases

A typical relational database is not a viable solution to store plant data because of the following limitations:

- Cannot handle the volume of data produced by plants.
- Cannot handle the rapid storage rate of plant data.
- SQL does not effectively handle time-series data.

Industrial plants have thousands of tags, all changing at different rates. Several months of plant history result in hundreds of gigabytes of data in a normal relational database.

For example, a plant with 10,000 variables changing on the average of every two seconds generates 5,000 values per second. 5,000 rows of data must therefore be inserted into the database each second to store a complete history, which is unsustainable by typical relational databases like Oracle or SQL Server on standard computer hardware.

Wonderware Historian as a Real-Time Relational Database

As a real-time relational database, Wonderware Historian is an extension to Microsoft SQL Server, providing more than an order of magnitude increase in acquisition speeds, a corresponding reduction in storage volume, and elegant extensions to structured query language (SQL) to query time series data.

- High-speed data capture

  The comprehensive range of Wonderware I/O Servers and DAServers are used to connect to over 500 control and data acquisition devices.

  Designed for optimal acquisition and storage of analog, discrete, and string data, the Wonderware Historian outperforms all normal relational databases on similar hardware by a wide margin, making the storage of high-speed data in a relational database possible. The historian acquires and stores process data many times faster than a RDBMS.

  Wonderware I/O Servers and DAServers support the SuiteLink™ protocol. SuiteLink allows for time and quality stamping at the I/O Server and further improves the rate of data acquisition.

- Reduced storage space

  The Wonderware Historian stores data in a fraction of the space required by a normal relational database. The actual disk space required to store plant data depends on the size and nature of the plant and the length of the plant history required.

- Time domain extensions to SQL

  The SQL language does not support time series data. In particular, there is no way to control the resolution of returned data in SQL. An example of resolution would be an evenly spaced sampling of data over a period of time.

  Microsoft SQL Server supports its own extensions to the SQL language, called Transact-SQL. The Wonderware Historian further extends Transact-SQL, allowing control of resolution and providing the basis for time-related functions such as rate of change and process calculations on the server.

Integration with Microsoft SQL Server

A large amount of plant-related data has the same characteristics as normal business data. For example, configuration data is relatively static and does not change at a real-time rate. Over the life of a plant, tags are added and deleted, descriptions are changed, and engineering ranges are altered. A Microsoft SQL Server database, called the *Runtime* database, stores this type of information.

The Runtime database is the SQL Server online database for the entire Wonderware Historian. The Runtime database is shipped with a set of standard database entities, such as tables, views, and stored procedures to store configuration data for a typical factory. You can use the Configuration Editor within the System Management Console to easily add configuration data to the Runtime database that reflects your factory environment.

Microsoft SQL Server Object Linking and Embedding for Databases (OLE DB) is used to access the real-time plant data that the historian stores outside of the SQL Server database. You can query the Microsoft SQL Server for both configuration information in the Runtime database and historical data on disk, and the integration appears seamless.

Because the historian is tightly coupled to and effectively extends a Microsoft SQL Server, it can leverage all of the features that Microsoft SQL Server has to offer, such as database security, replication, and backups.

Support for SQL Clients

The client/server architecture of Wonderware Historian supports client applications on the desktop, while ensuring the integrity and security of data on the server. This client/server architecture provides common access to plant and process data: real-time and historical data, associated configuration, event, and business data. The computing power of both the client and the server is exploited by optimizing processor intensive operations on the server and minimizing data to be transmitted on the network to improve system performance.

The gateway for accessing any type of information in the historian is the Microsoft SQL Server. Thus, any client application that can connect to Microsoft SQL Server can also connect to the historian.

Two categories of client applications can be used to access and retrieve information from the historian:

- Clients developed specifically to access data from historian. Wonderware provides a number of client tools to address specific data representation and analysis requirements. Third-party query tools that specifically support Wonderware Historian are also available. These client tools remove the requirement for users to be familiar with SQL and provide intuitive point-and-click interfaces to access, analyze, and graph both current and historically acquired time-series data.
- Any third-party query tool that can access SQL or ODBC data sources. Numerous commercial query and reporting tools are available that provide rich, user-friendly interfaces to SQL-based data. All client tools with an interface to Microsoft SQL Server or ODBC are suitable for historian data access and reporting.

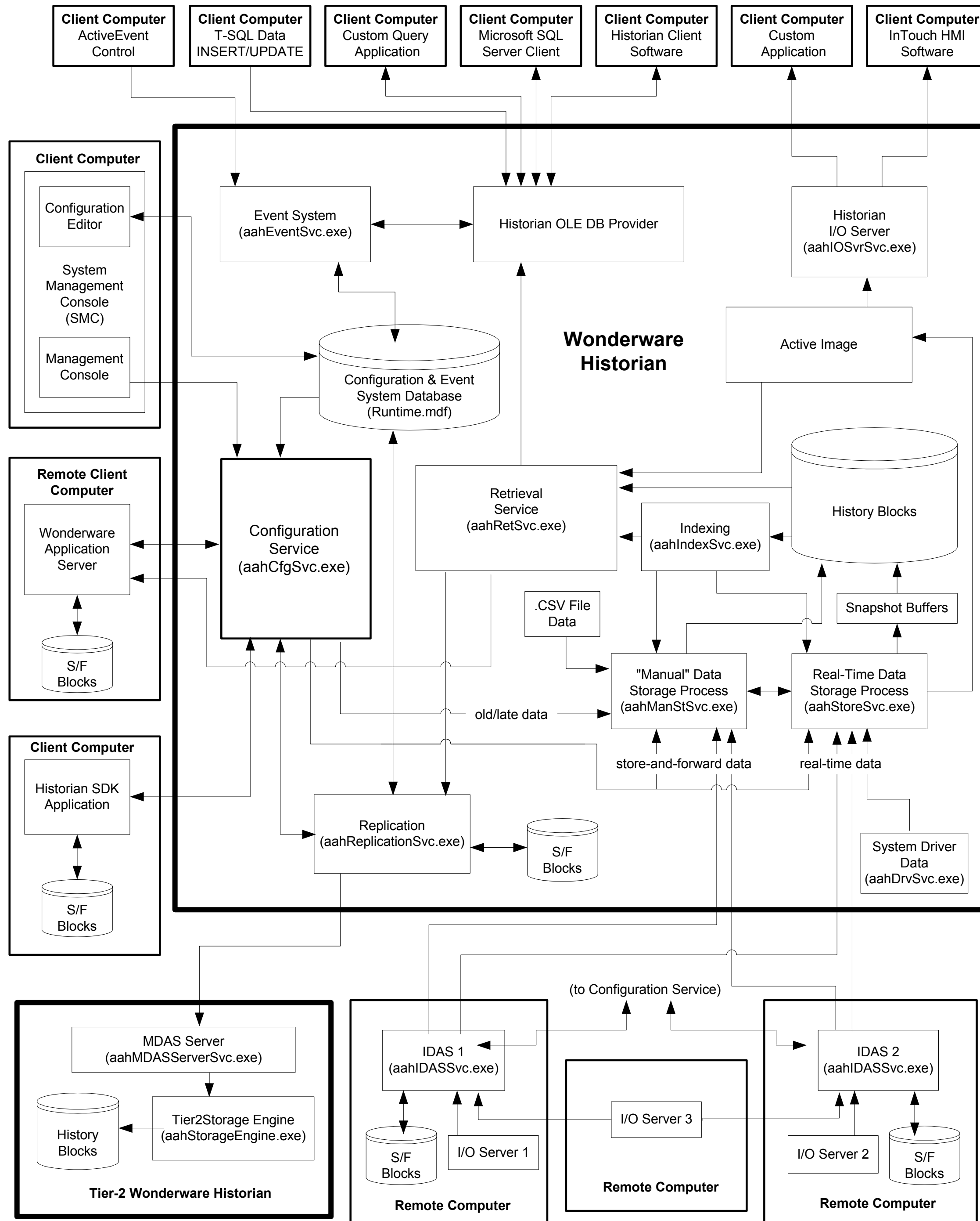
Wonderware Historian Subsystems

The Wonderware Historian is made up of specialized subsystems, which work together to manage data as it is acquired or generated, stored, and retrieved, as follows.

- Configuration Subsystem
- Data Acquisition Subsystem
- Data Storage Subsystem
- Data Retrieval Subsystem

- Event Subsystem
- Replication Subsystem

The following figure shows the overall architecture of the historian:



Chapter 2

System-Level Functionality

Some concepts apply across the entire Wonderware Historian, such as time handling, system parameters, security, and data quality.

About Tags

A tag is the atomic unit of storage in the Wonderware Historian. A tag is a variable that typically represents a single attribute of some physical process or device. A tag is characterized by a unique name in the historian. A tag has many attributes, such as type (for example, analog), how its values are acquired, or how its values are stored (cyclic or delta).

Types of Tags

The following table describes the types of tags in the system.

Tag Type	Description
Analog	An analog value is a variable that measures a continuous physical quantity. For example, the temperature of a boiler would be measured as an analog value.
Discrete	A discrete value is a variable that only has two states: '1' (True, On) or '0' (False, Off).
String	A string value is a text expression treated as a single data item. A string does not require a special format or syntax.

Tag Type	Description
Event	An event tag is a name for an event definition in the system. For example, if you wanted to detect when the temperature of tank reached 100 degrees, you might define an event tag and name it "TankAt100."
Analog Summary	An analog summary tag contains summarized data (minimum, maximum, average, and so on) that is configured to be replicated from one historian to another.
State Summary	An state summary tag contains summarized data (minimum time in state, maximum time in state, average time in state, and so on) that is configured to be replicated from one historian to another.

Sources of Tag Values

Sources for tag values are as follows:

- Automatically acquired I/O Server data, either in real-time or later.
- I/O Server data arriving late from an IDAS store-and-forward cache.
- Data generated internally for system monitoring tags.
- Data inserted or updated with a Transact-SQL statement.
- Data contained in a properly formatted .CSV file, which you import.
- Data sent from client applications developed with the Wonderware Historian Software Development Toolkit (SDK).
- Data from ArchestrA applications.
- Replicated data from one or more tier-1 Wonderware Historians

Naming Conventions for Tags

Tagnames may contain letters, digits, and special characters, where:

- letter = any letter as defined by the Unicode Standard. The Unicode definition of letters includes Latin characters from a through z and from A through Z, in addition to letter characters from other languages.
- digit = any numerical character
- special character = any graphics character except the following: characters whose ASCII table code is 0 through 32 -(non-graphic characters) and . + - * / \ = ( ) ` ~ ! ^ & @ [ ] { } | : ; ' , < > ? " space

It is highly recommended that you adhere to the rules for SQL Server identifiers as well.

For "conventional" tagnames the first character may be a:

- Letter
- Digit
- Dollar sign ($) or pound sign (#)

Subsequent characters may be:

- A digit, but then the tagname must contain at least one letter
- Any of the supported special characters.

Due to storage formatting requirements, you cannot use either a quotation mark (") or one or more single quotation marks (') at the beginning or at the end of a tag name.

Tag names that do not comply with these rules are regarded as "unconventional."

In a SQL query against a wide table, unconventional tag names must be delimited with brackets ([ ]), because the tag name is used as a column name. For more information, see "Using an Unconventional Tagname in a Wide Table Query" on page 281.

Security

The Wonderware Historian uses two security mechanisms:

- Windows operating system security
- Microsoft SQL Server security

For clients to access the historian, they must pass through both of these security levels. The historian Management Console (within the System Management Console) in particular adds an additional level of security checking to restrict access to functions affecting the state of the historian to only authorized users. Also, some of the historian components require Windows and SQL Server logins.

For more information on configuring user rights assignments for local security policies, see the Microsoft documentation.

For information on how to manage security, see Chapter 8, "Managing Security," in your *Wonderware Historian Administration Guide*.

Windows Operating System Security

To log on to Wonderware Historian as a client, the first thing a user must be able to do is to log on to the operating system on their computer. For the Windows operating system, a valid user account, consisting of a login ID (username) and password, is required to log on to the computer. This Windows user account can also be used to gain access to network resources in the domain.

SQL Server also requires authentication in order for clients to connect to it. You can use either Windows authentication or SQL Server authentication. For more information, see "SQL Server Security" on page 25.

Default Windows Login for Wonderware Historian Services

All of the modules in the Wonderware Historian, except for the Management Console and the Configuration Editor, run as Windows services, and therefore require a valid Windows user account (the ArchestrA administrative user account) to ensure proper operation. This ArchestrA user account is specified during installation.

The ArchestrA account must be a member of the local administrators group on the server hosting the historian, as well as on all computers hosting a remote IDAS.

You can change the ArchestrA user account by using the ArchestrA Change Network Account Utility.

WARNING! Changing the ArchestrA user account affects all ArchestrA components that run as services, not just historian services. If you change this account, you must restart the computer.

Do not configure historian services (including remote IDASs) to run under a specific user account. All services should be configured by default to log on to the Windows operating system using the LocalSystem account. The historian services will impersonate the ArchestrA user account, and this impersonation will fail if the service is not run using the LocalSystem account.

SQL Server Security

Because the Wonderware Historian contains an embedded Microsoft SQL Server, it uses and takes advantage of the security features that Microsoft SQL Server has to offer. The purpose of security for a SQL Server is to control who can access the server, access specific databases within a server, and perform certain actions within a database.

A database user must pass through two stages of security for the historian:

- Authentication, which validates the user's identity to the server itself.
- Database authorization, which controls the database(s) that user can access, as well as the types of actions that the user can perform on objects within the database.

User authentication and database authorization are managed from Microsoft SQL Server Management Studio.

To access information in the Wonderware Historian databases, users need to be granted access to the databases. The historian is shipped with pre-configured database roles and user accounts to serve as a starting point for your security model. Database roles, SQL Server login IDs, and user accounts are managed using the Microsoft SQL Server Management Studio.

Authentication

Microsoft SQL Server authenticates users with individual login account and password combinations. After the user's identity is authenticated, if authentication is successful, the user is allowed to connect to a SQL Server instance. There are two types of authentication:

- Windows authentication, in which users must connect to the SQL Server using a Windows user account (a Windows login ID and password that is provided during the Windows login session).
- SQL Server authentication, in which users must connect to the SQL Server using SQL Server login account (a SQL Server login ID and password).

SQL Server can operate in one of two security modes, which control the type of accounts that must be used for access to the server:

- Windows authentication mode. In this mode, the SQL Server only uses Windows authentication.
- Mixed mode. In this mode, the SQL Server uses both Windows authentication and SQL Server authentication. If the login name matches the Windows network username, then validation is handled by the Windows security mechanisms. If the user login name does not match the Windows network username, then Microsoft SQL Server security mechanisms are used.

SQL Server authentication is provided for backward compatibility only. Microsoft recommends that you use Windows authentication, when possible.

For more information about authentication, see your Microsoft SQL Server documentation.

Default Windows Security Groups

The following Windows security groups are created by default on the Wonderware Historian computer. Use these groups to assign different levels of database permissions to users.

- aaAdministrators
- aaPowerUsers
- aaUsers
- aaReplicationUsers

Each group is automatically configured to be a member of the SQL Server database role with the same name. For example, the aaAdministrators Windows security group is a member of the default aaAdministrators SQL Server database role. If you add Windows users to the aaAdministrators security group, they will automatically be given permissions of the aaAdministrators SQL Server database role.

Wonderware Historian Default Logins

When the Wonderware Historian is installed, default SQL Server logins are created that you can use for logging on to the historian from client applications. These default logins provide "out of the box" functionality in that you do not have to create logins to start using the system. The following table describes the pre-configured logins:

Login Name	Password	Description
aaAdmin	pwAdmin	A user who can access and modify all data and create objects. Cannot drop the database or truncate tables.
aaPower	pwPower	A user with full read access and the ability to create objects and modify the contents of the non-core tables.
aaUser	pwUser	A read-only user who can access all data, but cannot modify data or consume database resources.
aadbo	pwddbo	Database owner. Full permissions.

The default database for each of these logins is the historian Runtime database. This default security model is provided as a starting point for system security and is suitable for many types of installations.

These logins are valid if the Microsoft SQL Server is set to mixed mode security. If only Windows authentication is used, you must configure the access rights for each user.

Important Never use blank passwords for logins.

The following logins are provided for backward compatibility only. They will be deprecated in a future release. Do not use these logins.

Login Name	Password	Description
wwUser	wwUser	Same as aaUser.
wwPower	wwPower	Same as aaPower.
wwAdmin	wwAdmin	Same as aaAdmin.
wwdbo	pwddbo	Same as aadbo.

Database Authorization

After a user successfully connects to the Microsoft SQL Server, the user needs authority to access databases on the server. This is accomplished by user accounts for each database. A database user consists of a user name and a login ID. Each database user must be mapped to an existing login ID.

User names are stored in the sysusers table in each database. When a user tries to access a database, the Microsoft SQL Server looks for an entry in the sysusers table and then tries to find a match in the syslogins table in the master database. If the Microsoft SQL Server cannot resolve the username, database access is denied.

The types of actions the user can perform in the database are based on authority information defined in the user account. The authority to perform a certain action is called a permission. There are two types of permissions: object permissions and statement permissions.

Permission	Description
Object	Regulates the actions that a user can perform on certain database objects that already exist in the database. Database objects include things such as tables, indexes, views, defaults, triggers, rules, and procedures. Object permissions are granted and revoked by the owner (creator) of the object.

Permission	Description
Statement	Controls who can issue particular Transact-SQL statements. Database statements include commands such as SELECT, INSERT, or DELETE. Statement permissions, also called command permissions, can only be granted and revoked by the system administrator or the database owner.

Users can be grouped into roles, which is a single unit against which you can apply permissions. Permissions granted to, denied to, or revoked from a role also apply to any members of the role.

Wonderware Historian Default Users and Roles

The Wonderware Historian is shipped with a number of pre-configured user accounts and roles.

The following table describes the default SQL Server usernames, the login IDs and database roles to which they belong, and the actions that they are allowed to perform in the Runtime database. You can add additional users and roles using SQL Server Enterprise Manager.

Login ID	Username in Database	Member of Role	Permissions
aaUser	aaUser	aaUsers	SELECT on all tables INSERT, UPDATE, DELETE on PrivateNameSpace and Annotation
aaPower	aaPower	aaPowerUsers	CREATE Table CREATE View CREATE Stored procedure CREATE Default CREATE Rule SELECT on all tables INSERT, UPDATE, DELETE on grouping tables

Login ID	Username in Database	Member of Role	Permissions
aaAdmin	aaAdmin	aaAdministrators	CREATE Table CREATE View CREATE Stored procedure CREATE Default CREATE Rule DUMP Database DUMP Transaction SELECT, INSERT, UPDATE, DELETE on all tables
aadbo	dbo	db_owner	Full database owner capabilities

The following users and roles are provided for backward compatibility only. They will be deprecated in a future release. Do not use these users and roles.

Login ID	Username in Database	Member of Role	Permissions
wwUser	wwUser	wwUsers	Same as for aaUser.
wwPower	wwPower	wwPowerUsers	Same as for aaPower.
wwAdmin	wwAdmin	wwAdministrators	Same as for aaAdmin.
wwdbo	wwdbo	db_owner	Same as for aadbo.

Each default role contains the corresponding SQL Server user account, as well as the corresponding default Windows security group. For more information on the default Windows security groups, see "Default Windows Security Groups" on page 26.

Default SQL Server Login for Wonderware Historian Services

Some components of the Wonderware Historian require a SQL Server login ID to access the master, Runtime, and Holding databases. By default, the historian uses the ArchestrA user account to log on to the Microsoft SQL Server, using Windows authentication.

For Microsoft SQL Server, if the Windows user account is an administrative account on the local computer, it will map the account to the sysadmin fixed server role. (This user will be granted the same permissions as the **sa** SQL Server user account.) Because the ArchestrA user account is always a local administrative account, it will always have administrative permissions (sysadmin) within the SQL Server.

Management Console Security

The Wonderware Historian Management Console (which is part of the overall System Management Console) runs in the context of the logged on Windows user account. To protect against unauthorized access to the Wonderware Historian, you must specify a separate Windows user account that the Management Console will use to connect to the historian. You can specify this account when you set up the server registration properties. For more information on registration, see Chapter 1, "Getting Started with Administrative Tools," in your *Wonderware Historian Administration Guide*.

If the account specified is not a member of the local administrators group on the computer hosting the historian, the Management Console has "read-only" access. That is, you may view all the information shown in the Management Console, but you cannot perform any control actions on the historian, such as starting or stopping the system, creating new history blocks, and so on.

Important To prevent possible unauthorized access, the password for the Management Console login account must NOT be blank.

Default Access Rights for Different Operating Systems

By default, in the Windows XP operating system, all users are given the right to shut down the local computer. Therefore, for these operating systems, all users are automatically given "level 2" access to the historian. A "level 2" user in the Wonderware historian can shut down the local computer. To prevent users from being able to start or stop the historian on these operating systems, you must take away the "Shut Down the System" right in the Windows local security policy.

For the Windows Server 2003 operating system, "power users" are given the right to shut down the local computer. Power users are members of "Shut Down the System" right in the Windows local security policy. Therefore, for this operating system, power users are automatically given "level 2" access to the historian.

For the Windows Server 2008 and Windows Vista operating systems, "power users" are not given the right to shut down the local computer. Power users are not members of "Shut Down the System" right in the Windows local security policy. Therefore, for these operating systems, power users are automatically given "level 1" access to the historian.

Time Handling

Timestamps for all data are stored in Coordinated Universal Time (UTC), also known as Greenwich Mean Time. The current UTC time is derived from the current local time and the time zone setting in the operating system of the computer on which the Wonderware Historian is running. During data retrieval, timestamps are returned in local time, by default. You can convert the timestamps so that they are shown in local time by using a special query parameter.

You should use the international date/time format for all timestamps used in queries. The format is:

YYYYMMDD HH:MM:SS.000

where,

YYYY = year

MM = month

DD = day

HH = hour

MM = minutes

SS = seconds

000 = milliseconds

The format for timestamps returned from queries is controlled by the default language settings of the SQL Server login. Make sure that you configure the default language setting for SQL Server logins correctly, especially in environments with mixed languages/regional settings.

If you have multiple historians and/or are using remote data sources, it is very important that you synchronize the time between the computers. For more information, see "Time Synchronization for Data Acquisition" on page 87.

Make sure that you have selected the operating system setting to automatically adjust time for daylight savings, if the time zone in which the computer is located observes daylight savings time.

System Parameters

A system parameter is a parameter that controls some aspect of the overall Wonderware Historian behavior. The following table describes the default system parameters:

Name	Description
AIAutoResize	Controls the automatic resizing of the active image. 1 = Automatic resizing enabled; 0 = Automatic resizing disabled. The default is 1. For more information, see "About the Active Image" on page 119.
AIResizeInterval	Interval, in minutes, that the system will resize the active image, if the AIAutoResize parameter is enabled.
AllowOriginals	Used to allow the insertion of original data for I/O Server tags. You must set this parameter to 1 before importing .lgh original data. For more information, see "Data Acquisition by Means of INSERT and UPDATE Statements" on page 90.
AnalogSummaryTypeAbbreviation	Abbreviation used when generating analog summary tagnames. For more information, see "Specifying Naming Schemes for Replication" in Chapter 7, "Managing and Configuring Replication," in your *Wonderware Historian Administration Guide*.

Name	Description
AutoStart	Used to start the historian automatically when the computer on which the Wonderware Historian is running is started: 1 = Autostart enabled; 0 = Autostart disabled. For more information, see "Configuring Wonderware Historian to AutoStart" in Chapter 1, "Getting Started with Administrative Tools," in your *Wonderware Historian Administration Guide*. If you change this parameter, you must commit the changes to the historian system. **Note** You cannot change the Autostart system parameter using the SMC if SQL Server is not installed to run as an administrator. The SQL Server instance is used to change the service settings on behalf of the SMC.
ConfigEditorVersion	(Not editable.) The minimum version number of the Configuration Editor that can edit the Runtime database. Used internally by the system.
DatabaseVersion	(Not editable.) Current version number of the database.
DataImportPath	Path to the CSV file for an import of external data. For more information, see "Importing Data from a CSV File" on page 91. If you change this parameter, a restart of the system is required.
EventStorageDuration	Maximum duration, in hours, that event records are stored in the EventHistory table.
HeadroomXXXX	Number of tags for which to pre-allocate memory in the system. For more information, see "Pre-allocating Memory for Future Tags" in Chapter 2, "Configuring Tags," in your *Wonderware Historian Administration Guide*.
HistorianVersion	(Not editable.) Current version number and build number of the Wonderware Historian. The value for this parameter is automatically supplied when the system starts.

Name	Description
HistoryCacheSize	Allocation of system memory, in MB, for tag information. The default is 0. For more information, see "Memory Management for Data Storage" on page 122. If you change this parameter, you must commit the change and then rescan the history blocks to flush the cache.
HistoryDaysAlwaysCached	The duration, in days, for which history block information is always loaded in memory. The default is 0.
HoursPerBlock	Duration, in hours, for history blocks. Valid values are: 1, 2, 3, 4, 6, 8, 12, 24. The default is 24 hours. The history block size must always be greater than the highest scan rate. For more information, see "History Blocks" on page 113.
InterpolationTypeInteger	The type of interpolation for data values of type integer. 0=Stair-step; 1=Linear. The default is 0. For more information on interpolation, see "Interpolation Type (wwInterpolationType)" on page 237.
InterpolationTypeReal	The type of interpolation for data values of type real. 0=Stair-step; 1=Linear. The default is 1.
LateDataPathThreshold	Controls the store-and-forward threshold for late data.
LicenseRemoteIDASCount	(Not editable.) The number of allowed remote IDASs for the historian. This value is determined from the license file. Used internally by the system.
LicenseTagCount	(Not editable.) The number of allowed tags for the historian. This value is determined from the license file. Used internally by the system.
ManualDataPathThreshold	Controls the store-and-forward threshold for manual data.

Name	Description
ModLogTrackingStatus	Turns modification tracking on or off. The value you specify will determine what modifications are tracked. For more information, see "Turning Modification Tracking On/Off" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide*.
OldDataSynchronizationDelay	Time delay, in seconds, between when changes for “old” data (inserts, updates, and store-and-forward data) must be sent from the tier-1 historian to the tier-2 historian.
QualityRule	Indicates whether the system should use values having a quality of Good and Uncertain, or having only a quality of Good. 0 = Good and Uncertain; 1 = Good only. The default is 0. For more information on the quality rule, see "Quality Rule (wwQualityRule)" on page 244.
RealTimeWindow	The maximum delay, in seconds, for which data is considered real-time data for swinging door storage. The delay is relative to the current time. Valid values are between 30 and 300 milliseconds. The default is 60. For more information, see "About the Real-Time Data Window" on page 97.
ReplicationConcurrentOperations	Limits the total number of retrieval client objects performing calculations in a retrieval based calculations for a time cycle.
ReplicationDefaultPrefix	The default prefix for replication tags on the tier-2 historian. If you change ReplicationDefaultPrefix system parameter, all replication tags that use the old prefix are not updated to use the newer prefix. For more information, see "Specifying Naming Schemes for Replication" in Chapter 7, "Managing and Configuring Replication," in your *Wonderware Historian Administration Guide*.

Name	Description
ReplicationTCPPort	The TCP port number the tier-2 historian listens on for any incoming-connection requests from a tier-1 historian. It must match the port number the tier-1 is sending on for replication to succeed. When modifying this system parameter on a tier-2 historian node, you must also modify the port number in the Windows Firewall exception list for the historian replication service to the same value. This port number must unique on the tier-2 node; that is, no other applications on the tier-2 node should be listening on this port number.
RevisionLogPath	The file path to the write-ahead log for tier-2 insert/update transactions.
SimpleReplicationNamingScheme	The default naming scheme used for configuring simple replication tags. For more information, see "Specifying Naming Schemes for Replication" in Chapter 7, "Managing and Configuring Replication," in your *Wonderware Historian Administration Guide*.
StateSummaryTypeAbbreviation	Abbreviation used when generating state summary tagnames. For more information, see "Specifying Naming Schemes for Replication" in Chapter 7, "Managing and Configuring Replication," in your *Wonderware Historian Administration Guide*.
SuiteLinkTimeSyncInterval	Frequency, in minutes, that IDASs will attempt to synchnronize the timestamping mechanism for associated I/O Servers. If this parameter is set to 0, no time synchronization will occur. For more information, see "Time Synchronization for Data Acquisition" on page 87.
SummaryCalculationTimeout	The maximum expected delay, in minutes, for calculating summary data for replicated tags. Setting this parameter too high will delay associated summary calculations unnecessarily. Setting it too low will cause the system to prematurely calculate summaries and then later require additional processing to correct those calculations.

Name	Description
SummaryReplicationNamingScheme	The default naming scheme used for configuring summary replication tags. For more information, see "Specifying Naming Schemes for Replication" in Chapter 7, "Managing and Configuring Replication," in your *Wonderware Historian Administration Guide*.
SummaryStorageDuration	Maximum duration, in hours, that summary records will be stored in the legacy SummaryHistory table.
SysPerfTags	Used to turn on performance monitoring tags for the Wonderware Historian system. 0 = Off; 1 = On. The default is 1. For more information, see "Performance Monitoring Tags" on page 48.
TimeStampRule	Used to determine which timestamp within a retrieval cycle to use for a data value. 0 = Use the timestamp at the start of the cycle; 1 = Use the timestamp at the end of the cycle. The default is 1. For more information, see "Timestamp Rule (wwTimestampRule)" on page 240.
TimeSyncIODrivers	If enabled, the Wonderware Historian will send time synchronization commands to all associated remote IDASs. For more information, see "Time Synchronization for Data Acquisition" on page 87.
TimeSyncMaster	Name of the computer that the Wonderware Historian will use as a time synchronization source.

System Messages

System messages include error messages and informational messages about the state of the Wonderware Historian as a whole or for any of the internal subsystems and individual processes. System messages are logged to the:

- ArchestrA Logger.
- Windows event log, which can be viewed with the Windows Event Viewer. Not all messages are logged to the Windows event log. In general, only user actions and

exceptional events are written to this log. The messages are logged with the "Historian" or the name of the Wonderware Historian service as the source.

System messages are divided into the following categories:

Category	Description
FATAL	The process cannot continue. An error of this severity results in a system shutdown.
CRITICAL	These types of errors will cause malfunctions in the data storage or retrieval systems, such as data loss or corruption.
ERROR	General errors. For example, address validation errors during system startup. These errors may result in an orderly shutdown of the system, but will not preclude system operation in most cases.
WARNING	Messages that simply notify the operator of parameter settings or events that have occurred. For example, failure to link a dynamically-linked procedure entry point for a non-obligatory function will be logged as a warning.
INFO	Messages relating to startup progress or the commencement of active data storage.
DEBUG	Debugging messages, which will not typically appear in released versions of the system.

Wonderware Historian messages are logged to the Log Viewer as follows:

- Critical, fatal, and error messages are logged as "Error" messages. The appropriate indicator, either "CRITICAL," FATAL," or "ERROR," will be prefixed to message.
- Warnings will be logged as "Warning" message, with no prefix.
- Informational messages will be logged as "Info" messages, with no prefix.
- Debug messages will be logged as "Trace" messages, with no prefix.

For information on monitoring the system, see Chapter 10, "Monitoring the System," in your *Wonderware Historian Administration Guide*.

Wonderware Historian Services

The following Wonderware Historian processes run as Windows services:

Display Name (Service Name)	Executable Name	Description
Wonderware Historian Configuration (InSQLConfiguration)	aahCfgSvc.exe	Handles all configuration requests, as well as hosts the interfaces for manual data input and retrieval. For more information, see Chapter 3, "Configuration Subsystem."
Wonderware Historian DataAcquisition (InSQLDataAcquisition)	aahIDASSvc.exe	Acquires data from local or remote IDASs and forwards it on to the storage subsystem. For more information, see Chapter 4, "Data Acquisition Subsystem."
Wonderware Historian EventSystem (InSQLEventSystem)	aahEventSvc.exe	Searches through history data and determines if specific events have occurred. For more information, see Chapter 10, "Event Subsystem."
Wonderware Historian Indexing (InSQLIndexing)	aahIndexSvc.exe	Manages the indexing of history data on disk. For more information, see Chapter 5, "Data Storage Subsystem."
Wonderware Historian IOServer (InSQLIOServer)	aahIOSvrSvc.exe	Provides realtime data values from the historian to network clients. For more information, see Chapter 6, "Data Retrieval Subsystem."
Wonderware Historian ManualStorage (InSQLManualStorage)	aahManStSvc.exe	Accepts all incoming non-realtime plant data and stores it to disk. For more information, see Chapter 5, "Data Storage Subsystem."

Display Name (Service Name)	Executable Name	Description
Wonderware Historian MDASServer (aahMDASServer)	aahMDASServerSvc.exe	Manages data and communications between tier-1 and tier-2 historians. For more information, see Chapter 4, "Data Acquisition Subsystem."
Wonderware Historian Replication (HistorianReplication)	aahReplicationSvc.exe	Performs data transformations on a tier-1 historian and sends the results to one or more tier-2 historians. For more information, see Chapter 9, "Replication Subsystem."
Wonderware Historian Retrieval (InSQLRetrieval)	aahRetSvc.exe	Retrieves data from storage. For more information, see Chapter 6, "Data Retrieval Subsystem."
Wonderware Historian SCM (InSQLSCM)	aahSCM.exe	Provides status information regarding the historian. Used internally by the historian. This service runs continuously, even if the historian is stopped.
WonderwareHistorian Storage (InSQLStorage)	aahStoreSvc.exe	Accepts all incoming real-time plant data and stores it to disk. For more information, see Chapter 5, "Data Storage Subsystem."
WonderwareHistorian SystemDriver (InSQLSystemDriver)	aahDrvSvc.exe	Generates data values for various system monitoring tags. For more information, see "The System Driver and System Tags" on page 42.

For more information on Windows services, see your Microsoft documentation.

The System Driver and System Tags

The system driver is an internal process that monitors key variables within an operating Wonderware Historian and outputs the values by means of a set of system tags. The system driver runs as a Windows service and starts automatically when the storage system is started.

The system tags are automatically created when you install the historian. Also, additional system tags are created for each IDAS and replication server you configure.

The current value for an analog system tag is sent to the storage subsystem according to a specified rate, in milliseconds. All date/time tags report the local time for the historian.

Error Count Tags

The following analog tags have a storage rate of 1 minute (60000 ms). All error counts are since the Wonderware Historian is restarted or since or the last error count reset.

TagName	Description
SysCritErrCnt	Number of critical errors
SysErrErrCnt	Number of non-fatal errors
SysFatalErrCnt	Number of fatal errors
SysWarnErrCnt	Number of warnings

Date Tags

The following analog tags have a storage rate of 5 minutes (300000 ms).

TagName	Description
SysDateDay	Day of the month
SysDateMonth	Month of the year
SysDateYear	Four-digit year

Time Tags

All of the following tags are analog tags. Each value change is stored (delta storage).

TagName	Description
SysTimeHour	Hour of the day
SysTimeMin	Minute of the hour
SysTimeSec	Second of the minute

Storage Space Tags

The following analog tags have a storage rate of 5 minutes (300000 milliseconds). Space remaining is measured in MB.

TagName	Description
SysSpaceAlt	Space left in the alternate storage path
SysSpaceBuffer	Space left in the buffer storage path
SysSpaceMain	Space left in the circular storage path
SysSpacePerm	Space left in the permanent storage path

I/O Statistics Tags

The following analog tags can be used to monitor key I/O information.

TagName	Description
SysDataAcq*N*BadValues*	Number of data values with bad quality received. This tag has a storage rate of 5 seconds. The maximum is 1,000,000.
SysDataAcq*N*OutsideRealtime*	The number of values per second that were discarded because they arrived outside of the real-time data window. This tag has a storage rate of 5 seconds. The maximum is 1,000,000.
SysDataAcqOverallItemsPerSec	The number of items received from all data sources. This tag has a storage rate of 10 seconds. The maximum is 100,000.

TagName	Description
SysDataAcqRxItemPerSec*N**	Tag value update received per second. This tag has a storage rate of 10,000 milliseconds. Updated every 2 seconds for this IDAS.
SysDataAcqRxTotalItems*N**	Total number of tag updates received since last startup for this IDAS. This tag has a storage rate of 10,000 milliseconds.
SysStatusRxItemsPerSec	Tag value update received per second. Updated every 2 seconds for the system driver (aahDrvSvc.Exe). This tag has a storage rate of 1,000 milliseconds.
SysStatusRxTotalItems	Total number of tag updates received since last startup for the system driver. This tag has a storage rate of 10,000 milliseconds.
SysStatusTopicsRxData	Total number of topics receiving data.

*This status tag will exist for each defined IDAS. The identifying number (*N*) in the is the **IODriverKey** from the **IODriver** table. The number 0 designates MDAS and only applies to the SysDataAcq*N*BadValues and SysDataAcq*N*OutsideRealtime tags.

System Monitoring Tags

Unless otherwise noted, for the following discrete tags, 0 = Bad; 1 = Good.

Tag	Description
SysConfiguration	Status of the configuration service (aahCfgSvc.exe). This parameter is set to 1 as long as a dynamic configuration is required or in progress.
SysDataAcq*N**	Status of the IDAS service (aahIDASSvc.exe).
SysEventSystem	Status of the event system service (aahEventSvc.exe).
SysIndexing	Status of the indexing service (aahIndexSvc.exe).
SysInSQLIOS	Status of the Wonderware Historian I/O Server (aahIOSvrSvc.exe).
SysManualStorage	Status of the manual storage service (aahManStSvc.exe).
SysMDAServer	Status of the MDASServer service (aahMDASServerSvc.exe).

Tag	Description
SysOLEDB	Status of the OLE DB provider (loaded by SQL Server).
SysPulse	Discrete "pulse" tag that changes every minute.
SysReplication	Status of Replication service (aahReplSvc.exe).
SysRetrieval	Status of the retrieval service (aahRetSvc.exe).
SysStorage	Status of the storage service (aahStoreSvc.exe).
SysSystemDriver	Status of the system driver (aahDrvSvc.exe).
SysTier2Storage	Status of tier-2 storage.

*This status tag will exist for each defined IDAS. The identifying number (*N*) appended to the end of the tag is the **IODriverKey** from the **IODriver** table.

Miscellaneous (Other) Tags

The following table describes miscellaneous tags.

Tag	Description
SysConfigStatus	Number of database items affected by a dynamic configuration (that is, the number of entries in the ConfigStatusPending table when the commit is performed). This value is cumulative and not reset until the system is completely restarted.
SysHeadroom*XXX*	Used to monitor the number of "headroom" tags still available. Analog tags are divided by byte size: 2, 4, or 8 byte. For more information, see "Pre-allocating Memory for Future Tags" in Chapter 2, "Configuring Tags," in your *Wonderware Historian Administration Guide*.
SysHistoryCacheFaults	The number of history blocks loaded from disk per minute. The maximum value is 1,000. The storage rate for this analog tag is 60 seconds. For more information on the history cache, see "Memory Management for Data Storage" on page 122.

Tag	Description
SysHistoryCacheUsed	Number of bytes used for history block information. The maximum value is 3,000,000,000. The storage rate for this analog tag is 30 seconds.
SysHistoryClients	The number of clients that are connected to the Indexing service. The maximum value is 200. The storage rate for this analog tag is 30 seconds.
SysMinutesRun	Minutes since the last startup. The storage rate is 60000 milliseconds for this analog tag.
SysString	String tag whose value changes every hour
SysRateDeadbandForcedValues	The total number of values that were forced to be stored as a result of using a swinging door storage deadband. This number reflects all forced values for all tags since the system was started.

Event Subsystem Tags

The following table describes the event subsystem tags.

TagName	Description
SysEventCritActionQSize	Size of the critical action queue. For more information, see "Action Thread Pooling" on page 355.
SysEventDelayedActionQSize	Number of entries in the delayed action queue.
SysEventNormActionQSize	Size of the normal action queue.
SysEventSystem	A discrete tag that indicates the status of the event system service (aahEventSvc.exe). 0 = Bad; 1 = Good.
SysStatusEvent	Snapshot event tag whose value changes every hour.

Replication Subsystem Tags

The Replication Service collects the following custom performance counters about its own operation, where N is a primary key of the tier-2 historian in the Runtime database of the tier-1 historian. These values are stored cyclically every 10 seconds.

TagName	Description
SysReplicationSummaryCalcQueueItemsTotal	Current number of summary calculations stored in the summary calculation queue of all tier-2 historians.
SysReplicationSummaryClientsTotal	Current number of concurrent retrieval clients performing summary calculations on the tier-1 historian for all tier-2 historians.
SysReplicationSyncQueueItems*N*	Current number of items stored in the synchronization queue on the tier-2 historian of key *N*.
SysReplicationSyncQueueItemsTotal	Current number of items stored in the synchronization queue on the tier-1 for all tier-2 historians.
SysReplicationSyncQueueValuesPerSec*N*	Average synchronization queue values per second sent to the tier-2 historian of key *N*.
SysReplicationSyncQueueValuesPerSecTotal	Average values processed by the replication synchronization queue processor for all tier-2 historians.
SysReplicationTotalTags*N*	Total number of tags being replicated to the tier-2 historian of key *N*.
SysReplicationTotalValues*N*	Total number of values sent to the tier-2 historian of key *N* since the startup of the replication service.
SysReplicationTotalValuesTotal	Total number of values sent to all tier-2 historians since the startup of the replication service.
SysReplicationValuesPerSec*N*	Average values per second sent to the tier-2 historian of key *N*
SysReplicationValuesPerSecTotal	Average values per second sent to all tier-2 historians.

Performance Monitoring Tags

You use performance monitoring tags to monitor CPU loading and other performance parameters for various Wonderware Historian processes. (All of these values map to equivalent counters that are used in the Microsoft Performance Logs and Alerts application.)

The following tags allow you to monitor the percentage CPU load for each processor (up to a total of four), as well as the total load for all processors:

- SysPerfCPU0
- SysPerfCPU1
- SysPerfCPU2
- SysPerfCPU3
- SysPerfCPUTotal

The remaining system tags are used to monitor performance for each historian process that runs as a Windows service and for the Microsoft SQL Server service. For more information on services, see "Wonderware Historian Services" on page 40.

There are six system performance tags per each service. These tags adhere to the following naming convention, where *XXX* designates the service (Config, DataAcq, EventSys, Indexing, InSQLIOS, ManualStorage, MDASServer, Replication, Retreival, SQLServer, Storage, SysDrv, or Tier2Storage):

SysPerf*XXX*CPU

SysPerf*XXX*HandleCount

SysPerf*XXX*PageFaults

SysPerf*XXX*PrivateBytes

SysPerf*XXX*ThreadCount

SysPerf*XXX*VirtualBytes

These tags have a cyclic storage rate of 5 seconds.

Note The six performance tags will exist for each defined IDAS. The identifying number (*N*) appended to the end of the "DataAcq" portion of the tagname is the **IODriverKey** from the **IODriver** table. For example, 'SysPerfDataAcq1CPU'.

The following table describes the suffixes assigned to the names of system performance tags:

Suffix	Description
CPU	Current percentage load on the service.
HandleCount	Total number of handles currently open by each thread in the service. A handle is a identifier for a particular resource in the system, such as a registry key or file.
PageFaults	Rate, per second, at which page faults occur in the threads executing the service. A page fault will occur if a thread refers to a virtual memory page that is not in its working set in main memory. Thus, the page will not be fetched from disk if it is on the standby list (and already in main memory) or if it is being used by another process.
PrivateBytes	Current number of bytes allocated by the service that cannot be shared with any other processes.
ThreadCount	Current number of active threads in the service. A thread executes instructions, which are the basic units of execution in a processor.
VirtualBytes	Current size, in bytes, of the virtual address space that is being used by the service.

Supported Protocols

Wonderware Historian supports the following protocols:

Protocol	Description
DDE	DDE is the passage of data between applications, accomplished without user involvement or monitoring. In the Windows environment, DDE is achieved through a set of message types, recommended procedures (protocols) for processing these message types, and some newly defined data types. By following the protocols, applications that were written independently of each other can pass data between themselves without involvement on the part of the user. For example, InTouch HMI software and Excel.
SuiteLink	SuiteLink is a protocol that provides substantially more application level functionality than that provided by DDE. The SuiteLink protocol allows for the passing of time stamp and quality information with process data in the data packets.
System protocol	The system protocol in an internal means of passing data from system variables to the historian.

Note DDE is not supported if the historian is running on the Windows Server 2003, Windows Server 2008, or Windows Vista operating system.

Modification Tracking

Wonderware Historian tracks modifications (inserts, updates, and deletions) to columns in the Runtime database. If your plant tracks changes for compliance with regulatory agencies, you can configure the historian to use modification tracking.

Modification tracking is system-wide; it is controlled by the ModLogTrackingStatus system parameter. You cannot turn modification tracking on or off at a table level. Enabling modification tracking decreases the historian's performance when making changes to the system. This is due to the extra work and space required to track the changes. However, there is no performance degradation during run-time operation.

Information in the modification tracking tables are stored in the data files of the Microsoft SQL Server database. If modification tracking is turned on, the amount of data that is stored in these files is greatly increased.

All of the objects for which modifications can be tracked are stored in the HistorianSysObjects table.

You can track two types of modifications:

- Changes to configuration data. For example, additions or changes to tag, I/O Server, and storage location definitions. For more information, see "Modification Tracking for Configuration Changes" on page 52.
- Changes to history data. For example, data inserts and updates by Transact-SQL statements or CSV imports. For more information, see "Modification Tracking for Historical Data Changes" on page 53.

The types of changes that will be tracked is controlled by the ModLogTrackingStatus system parameter. You can track inserts, updates, and deletes, as well as various combinations. For more information, see "Turning Modification Tracking On/Off" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide*.

Modification Tracking for Configuration Changes

For configuration data, when a modification is made to a table in the database, a record for the modification is inserted into the ModLogTable table. One row will be inserted for each separate type of modification, either an insert, update, or delete.

The actual value changes are recorded in the ModLogColumn table. Each column that is modified will result in a row inserted into the ModLogColumn table. The entry in the ModLogColumn table includes both the column value before the change and the new column value.

For example, if you added (inserted) a single analog tag to the system, the following changes would be reflected in the modification tracking tables:

- Two rows would be added to the ModLogTable table, one to track the change to the Tag table and one to track the change to the AnalogTag table.
- One row for each of the columns in both of the Tag and AnalogTag tables will be added to the ModLogColumn table.

As another example, if you updated for a single analog tag the StorageType column in the Tag table and the ValueDeadband and RateDeadband columns in the AnalogTag table, the following changes would be reflected in the modification tracking tables:

- Two rows would be added to the ModLogTable table, one to track the change to the Tag table and one to track the change to the AnalogTag table.
- Three rows would be added to the ModLogColumn table to record the changes to the StorageType, ValueDeadband, and RateDeadband columns.
- Important things to note:

For a tier-2 historian, modification tracking for a replicated tag appears as the being made by the system account that the configuration service is running under, which is typically NT AUTHORITY\SYSTEM. To find out who modified a tag, examine the ModLogTable of the tier-1 historian.

Modification Tracking for Historical Data Changes

Modifications to history data can be performed by either executing Transact-SQL statements or by using the CSV import functionality. In the case of Transact-SQL statements, the Wonderware Historian OLE DB provider provides the change information to the modification tracking tables by means of a stored procedure. This stored procedure is also used by the storage subsystem to communicate changes that are the result of a CSV import.

Although the history data that is changed is physically stored on disk in the history blocks, for the purposes of modification tracking, the data is considered to reside in the History_OLEDB extension table. For more information on extension tables, see "Extension (Remote) Tables for History Data" on page 132.

When a modification is made to history data, a record for the modification is inserted into the ModLogTable table. One row will be inserted for each separate type of modification, either an insert or an update, for each tag.

The ModLogColumn table is used to store details for the column modification in the History_OLEDB table. The modified column will always be the vValue column. The total count of consecutive value changes attempted per tag is stored in the NewValue column of the ModLogColumn table.

The OldValue column contains the value stored in the column before the modification was made, if the modification was to a configuration table. For modifications to history data using SQL INSERT and UPDATE statements, this column contains the timestamp of the earliest data affected by the INSERT or UPDATE operation. If multiple changes are made to the same data, then only the most recent change will be contained in this column. This column is not used for modifications made to history data using a CSV file.

For example, if you insert 20 data values into history for the ReactTemp analog tag using a CSV import, the following changes would be reflected in the modification tracking tables:

- One row would be added to the ModLogTable table, to track the change to the History_OLEDB table. The UserName column will contain the name of the user as contained in the CSV file header.
- One row would be added to the ModLogColumn table to record that the value change occurred. A value of 20 will be stored in the NewValue column to indicate that 20 values were inserted.

Data Quality

Data quality is the degree of validity for a data value. Data quality can range from good, in which case the value is exactly what was originally acquired from the plant floor, to invalid, in which the value is either wrong or cannot be verified. As a data value is acquired, stored, retrieved, and then shown, its quality can degrade along the way, as external variables and events impact the system. There are three aspects of quality handling for the system:

- Data quality assignment by data acquisition devices (OPCQuality)
- Storage subsystem quality for the Wonderware Historian (QualityDetail)
- Client-side quality definitions (Quality

The historian uses three distinct parameters to indicate the quality of every historized data value: Quality, QualityDetail, and OPCQuality. OPCQuality is strongly related to QualityDetail. Quality is a derived measure of the validity of the data value, while QualityDetail and OPCQuality indicate either a good quality for the data value, or the specific cause for degraded quality of the data value.

The historian persists four bytes of quality information for every data value that is historized. (This does not imply that four bytes of quality data are actually stored for every data value, but it guarantees that every data value can be associated with 4 bytes of quality information when the value is retrieved.) The 4 bytes of quality information comprises 2 bytes for QualityDetail and 2 bytes for OPCQuality. QualityDetail and OPCQuality are related, but may have different values.

In essence, OPCQuality is provided as a migration path for future client applications that are fully aware of the OPC data quality standard, while QualityDetail (and Quality) are maintained to ensure proper operation of existing and legacy client applications. OPCQuality is sent by the data source, and QualityDetail is added by the Wonderware Historian.

Viewing Quality Values and Details

The currently supported OPCQuality values are stored in the OPCQualityMap table of the Runtime database. To view OPCQuality values and descriptions, execute the following query:

```
SELECT OPCQuality, Description FROM OPCQualityMap
```

The currently supported QualityDetail values are stored in the QualityMap table of the Runtime database. To view the complete list of QualityDetail values and descriptions, execute the following query:

```
SELECT QualityDetail, QualityString FROM QualityMap
```

The following notes apply to the QualityDetail values:

- The boundary point for a QualityDetail of 448 is 1 second.
- Although a quality detail of 65536 is used to indicate block gaps for tier-2 tags, NULL values are not produced for block gaps for tier-2 tags.

Basic QualityDetail Codes

Quality detail codes are metadata used to describe data values stored or retrieved using the Wonderware Historian. Quality details are 2 byte values that use the low order byte for the basic quality detail and the high order byte to store quality detail flags. The basic quality detail and quality detail flags are OR'd together.

The following table lists the basic quality detail codes. These codes use the low order byte, and the individual bit values do not have independent significance. For information on how to retrieve the complete list of quality detail codes, see Viewing Quality Values and Details on page 55.

QualityDetail	Quality String	Description	Created By
0	Bad Quality of undetermined state	Bad quality.	Storage
1	No data available, tag did not exist at the time	For cyclic, delta, full, interpolated, and best-fit, retrieval started before the tag was created (NULL value).	Retrieval
2	Running insert	Not used in IndustrialSQL Server 8.0 or later. Obsolete.	--
10	Communication loss	Pipe disconnect, goes with 248.	Storage
16	Good value, received out of time sync (cyclic tag)	Not used in IndustrialSQL Server 8.0 or later. Obsolete.	--
17	Duplicate time stamp; infinite slope	Slope calculation of a vertical line.	Retrieval
20	IDAS overflow recovery	An IDAS data buffer overflow was detected, and all overflow values were discarded to recover.	IDAS
24	IOServer communication failed	I/O Server communication failure. Application Server sends quality of 24 or 32.	Storage, MDAS
33	Violation of History Duration license feature	You have queried beyond the licensed time range limit.	Retrieval

QualityDetail	Quality String	Description	Created By
44	Pipe reconnect	First value received after start up or pipe reconnect. MDAS sends QualityDetail 44 for slow changing tags after fail-over.	Storage, MDAS
64	Cannot convert	The calculated point was based on points of different QualityDetails: Good, Doubtful, or Bad. This is, for example, used in the integral and average retrieval modes.	Retrieval
150	Storage startup	Storage startup.	Storage
151	Store forward storage startup	Store-and-forward storage startup.	Storage
152	Incomplete calculated value	Incomplete calculated value.	Replication, Storage
192	Good	Good value.	Storage, MDAS
202	Good point of version Latest	Current value is the latest update which “masks” a previous original value.	Storage
212	Counter rollover has occurred	Counter count reached the roll-over value.	Retrieval
248	First value received in store forward mode	First value received from pipe re-connect, delta retrieval, goes with QualityDetail 10. MDAS injects QualityDetail of 248 with the first value into the store-and-forward engine.	Storage, MDAS
249	Not a number	Value received was not a number, infinite value (NaN).	Storage, MDAS
252	First value received from IOServer	First value received after an I/O Server reconnect.	Storage

QualityDetail	Quality String	Description	Created By
65536	No data stored in history, NULL	NULL value. **Note** Although this quality detail is used to indicate block gaps on Tier 1 tags, NULL values are not produced for block gaps for Tier 2 tags.	Retrieval

QualityDetail Flags

The following bit flags are stored in the high order byte of the quality detail. Each bit is a flag that carries specific meaning.

0x0000 - Basic QualityDetail.

The following hi-byte flags are or'ed with the basic QualityDetail. The three flags cannot coexist; only one is set at any point in time.

0x0100 - Value received in the future.

0x0200 - Value received out of sequence.

0x0400 - Configured for server time stamping.

The following hi-byte flag can coexist with any of the prior four combinations.

0x0800 - Point generated by Rate Deadband filter.

The following hi-byte flag can coexist with any of the prior eight combinations.

0x1000 - Partial cycle, advance retrieval results.

The following hi-byte flag can coexist with any of the prior eight combinations.

0x2000 - Value has been modified by filter.

This bit is set during retrieval to indicate that retrieval has modified a value or time stamp due to applying a filter such as SnapTo() or ToDiscrete().

QualityDetail Bit Layout

The QualityDetail bit layout is as follows:

```
+----------------------+------------------------------+
|     Upper Byte       |              Lower Byte      |
13  12  11  10   9   8   7   6   5   4   3   2   1   0
 |   |   |   |   |   |   +- Basic QualityDetail
 |   |   |   |   |   +----- Value received in the future
 |   |   |   |   +--------- Value received out of sequence
 |   |   |   +------------- Configured for server time stamping
 |   |   +----------------- Point generated by rate deadband filtering
 |   +--------------------- Partial cycle, advanced retrieval
 +------------------------- Value modified by a filter
```

Acquisition and Storage of Quality Information

When a data value is acquired for storage in the Wonderware Historian, it is usually accompanied by a quality indicator generated at the source of the data. In general, the historian passes the source-supplied quality indicator to the storage subsystem unmodified, but there are instances where the data acquisition subsystem actually modifies the quality of the data value.

Quality for Data Acquired from I/O Servers

A data value acquired from a Wonderware I/O Server by an IDAS always has 2 bytes of quality information attached to it by the I/O Server. The IDAS presents the 2 bytes of quality information supplied by the I/O Server to the Wonderware Historian storage subsystem as OPCQuality, and the QualityDetail is set to 192. An exception to this is if the quality value sent by the I/O Server is 24, in which case both OPCQuality and QualityDetail are set to 24.

However, in some instances, IDAS will overwrite QualityDetail with a special value to indicate a specific event or condition. For more information on these values, see "Viewing Quality Values and Details" on page 55. When IDAS overwrites QualityDetail with one of the reserved values, it does not modify the OPCQuality value. Thus, QualityDetail contains the reserved value set by IDAS, while OPCQuality retains the value provided by the data source.

The SysDataAcq*N*BadValues system tag tracks the number of data values with bad quality that are coming from an IDAS. If this value is high, you should make sure that there is not problem with the IDAS or I/O Server. For more information on system tags, see "The System Driver and System Tags" on page 42.

Quality for Data Not Acquired from I/O Servers

The Wonderware Historian supports data acquisition from a variety of sources other than Wonderware I/O Servers. These sources include properly formatted CSV files, SQL queries, and the Wonderware Application Server.

Quality for data acquired from these sources is handled the same as for data received from Wonderware I/O Servers, with a few additional exceptions:

- For data acquired from CSV files, the specified quality is always interpreted as OPCQuality, and the QualityDetail is set to 192, unless the specified quality is 24, in which case both OPCQuality and QualityDetail are set to 24.
- If MDAS is responsible for acquiring the data (as is the case for SQL queries and Wonderware Application Server), the quality value presented by the source is preserved in OPCQuality, and QualityDetail is set to 192. Exceptions to this are:
  - If a quality value of 32 presented by the source to MDAS, OPCQuality is set to 32 and QualityDetail is set to 24.
  - If the data value presented by the source is infinite or NaN (not a number), OPCQuality contains the actual quality presented by the source, and QualityDetail is set to 249.
  - If a special condition or event occurs, MDAS will substitute a reserved value for QualityDetail. For more information, see "Viewing Quality Values and Details" on page 55.

Client-Side Quality

Three quality indicators are exposed to clients:

- A 1-byte short quality indicator (Quality)
- A 4-byte long quality detail indicator (QualityDetail)
- A 4-byte long OPC quality indicator (OPCQuality)

Quality contains summary information that essentially falls into the categories of Good, Bad, Doubtful, or InitialValue. This implicit information is derived from the data source as well as the Wonderware Historian.

Quality is a 1-byte indicator that provides a summary meaning for the related data item. Quality is an enumerated type with the following values:

Hex	Dec	Name	Description
0x00	0	Good	Good value.
0x01	1	Bad	Value marked as invalid.
0x10	16	Doubtful	Value is uncertain.
0x85	133	Initial Value (Good)	Initial value for a delta request.

Initial Value and Doubtful are derived from QualityDetail. The Initial Value is dependent on the type of query that is executed. For more information, see "Using Comparison Operators with Delta Retrieval" on page 294, "Using Comparison Operators with Cyclic Retrieval and Cycle Count" on page 299, and "Using Comparison Operators with Cyclic Retrieval and Resolution" on page 302.

QualityDetail contains detailed information that is potentially specific to the device which supplied it. Quality values may be derived from various sources, such as from I/O Servers or from the Wonderware Historian storage system.

To retrieve a listing of the quality values that are used by the historian, see "Viewing Quality Values and Details" on page 55.

Chapter 3

Configuration Subsystem

Configuration data is information about elements that make up the Wonderware Historian, such as tag definitions, I/O Server definitions, and storage locations for historical data files. Configuration data is relatively static and does not change frequently during normal plant operation. The configuration subsystem stores and manages configuration data.

Setting up the required databases and included entities (such as tables, stored procedures, and views) to support a typical factory environment would take countless hours. However, when you install the historian, all of these entities are defined for you, allowing you to quickly start using Wonderware Historian.

Configuration data is stored in SQL Server tables in the Runtime database. If you are already using InTouch® HMI software, you can easily import much of this information from existing InTouch applications, thus preserving your engineering investment. If you are using Application Server, much of the Wonderware Historian configuration is handled automatically by Application Server. You can also use the System Management Console to manually add definitions and configure the system. You can make bulk modifications to your historian configuration or migrate the configuration from one historian to another using the Wonderware Historian Database Export/Import Utility.

You can reconfigure the system at any time with no interruption in the acquisition, storage, and retrieval of unaffected tags. Configuration data can be stored with a complete revision history.

For information on how to configure your system, see Chapter 1, "Getting Started with Administrative Tools," in your *Wonderware Historian Administration Guide*.

Configuration Subsystem Components

The components of the configuration subsystem are:

Component	Description
Runtime database	SQL Server database that stores all configuration information.
Configuration and management tools	Consists of the System Management Console client application, the Wonderware Historian Database Export/Import Utility, and the configuration tools shipped with Microsoft SQL Server. For more information, see Chapter 1, "Getting Started with Administrative Tools," in your *Wonderware Historian Administration Guide*.
Configuration Service (aahCfgSvc.exe)	Internal process that handles all status and configuration information throughout the system. This process runs as a Windows service.

For a complete diagram of the Wonderware Historian architecture, see "Wonderware Historian Subsystems" on page 19.

About the Runtime and Holding Databases

A relational database management system (RDBMS) such as Microsoft SQL Server can contain many databases. A database is a collection of objects such as:

- Tables
- Stored procedures
- Views
- User-defined data types
- Users and groups

The Wonderware Historian is shipped with two pre-configured databases: the *Runtime* and *Holding* databases.

The historian embeds a full-featured Microsoft SQL Server. The historian supports all system tables associated with SQL Server. For more information on the Microsoft SQL Server tables, see your Microsoft documentation.

Note When installed on a case-sensitive Microsoft SQL Server, the Runtime and Holding databases are case-sensitive. Be sure that you use the correct case when performing queries.

The Runtime Database

The Runtime database is the online database against which the Wonderware Historian runs. The tables within the Runtime database store all configuration information, such as:

- System configuration.
- Tag definitions.
- InTouch integration information.
- System namespaces and grouping information.
- Event configuration information.
- User-entered annotations.

Runtime database tables are usually used as references or lookup tables by the historian and client applications. Any changes to the historian are reflected in these configuration tables. The configuration tables exist as normal SQL Server tables and data within them can be modified by using the Microsoft Transact-SQL query language. For more information on Transact-SQL, see your Microsoft documentation.

The Runtime database also stores some types of history data:

- Modification tracking data
- Event subsystem data

Tables that store modification tracking and event data are also normal SQL Server tables.

Finally, the Runtime database is used to **logically** store historized tag values. Although the tag values are stored in the history block files on disk, the values appear to be saved to tables in the Runtime database. For more information on history blocks, see "History Blocks" on page 113. For more information on retrieving historized tag values, see Chapter 6, "Data Retrieval Subsystem."

Note You cannot change the name of the Runtime database.

The Holding Database

The Holding database temporarily stores topic and configuration data imported into Wonderware Historian from an InTouch node. When you import configuration data from an InTouch application, the data is first mapped to table structures in the Holding database. Then, the data is moved into the Runtime database.

Important Do not modify any entities in the Holding database.

For more information about importing configuration information from an InTouch application, see Chapter 3, "Importing and Exporting Configuration Information," in your *Wonderware Historian Administration Guide*.

About the Configuration Service

The Configuration Service is an internal process that accepts configuration changes and updates the Runtime database. Thus, the Configuration Service is the only component that interacts with the configuration store.

The Configuration Service runs as a Windows service and accepts and distributes configuration information to and from different parts of the system by a set of interfaces. The Configuration Service also serves as a gateway for all information pertaining to the status of the different components of the Wonderware Historian.

Dynamic Configuration

The Wonderware Historian supports dynamic configuration; that is, you can modify the configuration of tags and other objects in the historian database while the system is running. The historian automatically detects and applies the modifications to its internal run-time state without requiring the system to be restarted. In addition, clients do not experience interruptions due to configuration changes.

The dynamic configuration feature in the historian caters for all possible database modifications that affect the run-time operation of the system. For some types of configuration modifications, the system automatically creates a new history block. The configuration subsystem is designed to ensure that no loss of data occurs for tags that are not affected by the modifications being applied. However, tags that require a change in data acquisition configuration will obviously lose data during the reconfiguration.

In most cases, the system continues to run uninterrupted. In the following cases, a restart of the system is required:

- When you change the main historization path in the system, a parameter that is rarely modified after installation.
- When you modify the DataImportPath system parameter.

For a description of the effect of various types of modifications made while the system is running, see "Effects of Configuration Changes on the System" on page 68.

Dynamic configuration is usually a two-step process:

1. Add, modify, or delete one or more objects in the database, using the System Management Console, Transact-SQL statements, or the database modification tool of your choice.

   As soon as you make a change, the Runtime database is updated to reflect the change. For example, when you add an analog tag using the wizard within the Configuration Editor, the database is updated as soon as you click **Finish**.

2. After making all of the modifications, you must commit the changes, which triggers the dynamic configuration process in the server. Modifications made to the system are done in a transactional fashion.

   The database modifications are not reflected in the running system until you commit the changes. You are committing changes to the system, not to the database.

You can commit changes to the configuration of the system as often as you want. You can also commit changes in batches or individually. There is no limit on the number of changes that may be committed to the system. Configuration changes typically take effect within 10 seconds under maximum data throughput conditions, unless a new history block is required, in which case the changes take longer (3 to 5 minutes) to take full effect.

For information on cases in which a commit is prevented, see "Cases in Which Configuration Changes are not Committed" on page 70.

Effects of Configuration Changes on the System

Different types of dynamic changes to the database affect the system in different ways.

Some types of modifications require a new history block to be created. To reduce the need to create new history blocks, the system includes "headroom" tags, which allocate extra space in the blocks to accommodate new tags. For more information on headroom tags, see "Pre-allocating Memory for Future Tags" in Chapter 2, "Configuring Tags," in your *Wonderware Historian Administration Guide*.

A summary of typical changes and their effect on the system after a commit is as follows.

- **Modifying system parameters**

  A modification to system parameters usually takes effect immediately (a new history block will not be created). The exception is adding headroom for one or more tag types, which requires a new history block. Also, if you change the HistoryCacheSize parameter and commit the change, the cache is not immediately flushed to bring the cache size to less than or equal to the new value. You must rescan the history blocks to flush the cache.

- **Modifying storage locations**

  Modifying the circular storage location requires a shutdown and restart of Wonderware Historian. Changes to the other storage locations take effect immediately.

- **Adding, deleting, and modifying tags**

  Adding one or more tags to the system generally results in a new history block being created, unless sufficient headroom is available for that particular tag type, in which case a new block is not required. If the headroom is exceeded, a new block is created and the headroom is replenished to the amount specified in the SystemParameter table.

  Deleting one or more tags takes effect immediately.

  Certain modifications to tags result in a new history block. Those include changing the integer size, changing the raw type, changing strings tags from fixed length to variable length or vice versa, changing storage type from "Not stored" to "Stored," changing a string tag from ASCII to Unicode or vice versa, and changing tag acquistion type from"IOServer" to "Manual" or vice versa.

  As a general guideline, modifications to a tag that changes its footprint on disk will result in a new history block. If only data acquisition or retrieval characteristics of a tag are modified, the changes take effect without requiring the system to create a new history block.

  Modifying data acquisition characteristics of a tag could result in a brief period of data loss, for that tag. As a guideline, any change to the source of data for the tag (for example, modifying the item name, topic name, or I/O server name of the tag) will result in a short gap in data for the tag, while the system disconnects from the old data source and connects to the new data source.

- **Adding, deleting, and modifying IDASs**

  Adding a new IDAS to the system results in a new set of system tags being added (the status and performance system tags associated with that IDAS). While adding an IDAS in itself does not require a new history block, the new system tags will result in a new block, unless sufficient headroom is available.

  Deleting an IDAS takes effect immediately. Modifying an IDAS never requires a new history block, but may result in data loss for the tags serviced by that IDAS (for example, moving an IDAS to another computer causes a disconnect from the data sources).

- **Adding, deleting, and modifying I/O Servers and topics**

  Adding or deleting I/O Servers and/or topics does not require a new history block. Modifying I/O Server or topic characteristics may result in data loss for their tags, if the modification implies a disconnect from the data source.

Cases in Which Configuration Changes are not Committed

If the system is not running, or storage is stopped, any commit is ignored and the contents of the ConfigStatusPending table are cleaned up. The exceptions are changes to the following fields in the SystemParameter table: HistoryCacheSize, HistoryDaysAlwaysCached, and AutoStart.

If the system is running, a commit will be disallowed:

- While a previous dynamic configuration is still in progress.

- While a new history block is in progress (whether the block is the result of a scheduled block changeover, a dynamic configuration, or a user request). A block changeover is in progress for five minutes after it has been created, and also ten minutes before the next scheduled block changeover.

For each case, a message appears, indicating that the commit is disallowed.

Chapter 4

Data Acquisition Subsystem

The Wonderware Historian has been designed for high-speed acquisition of data, acquiring and storing process data many times faster than a traditional relational database.

Wonderware Application Server, DA Servers, and I/O Servers are the main sources of plant data. The historian can acquire data from over 500 Wonderware and third-party I/O Servers, ensuring access to the industry's most comprehensive list of data acquisition and control devices. I/O Servers that use the SuiteLink protocol can provide time and quality stamping at the I/O Server level. Data can be acquired simultaneously from multiple I/O Servers over a variety of physical links, with a remote store-and-forward capability to prevent data loss in the event of failed network connection.

Custom client applications can be another source for real-time historical data. Clients that are developed with the Wonderware Historian Manual Data Acquisition Service (MDAS) can send historical tag values directly to the system.

You can batch import historical data formatted in a comma-separated values (CSV) file, allowing you to migrate existing data from other historians. You can also use the InTouch History Importer to easily import history data from InTouch HMI applications.

Finally, the historian generates data for key internal status variables, which allow you to monitor the health of the system.

Data Acquisition Components

The following table describes the components of the data acquisition subsystem. Many of the components run as Windows Services.

Component	Description
I/O Server (DAServer)	Wonderware-compatible software application that reads values from PLCs and other factory devices and forwards the real-time data to Wonderware applications.
IDAS Service	Process that accepts real-time data from one or more I/O Servers and forwards it to a single Wonderware Historian.
Query Tools	Any database query tool capable of issuing Transact-SQL INSERT or UPDATE statements. For example, Microsoft SQL Server Query Analyzer.
Data Import Folder	Defined file folder to batch import tag values to the historian.
InTouch History Importer	Utility to import data from one or more InTouch history files (.lgh). For more information, see "Importing Data from an InTouch History File" in Chapter 6, "Importing, Inserting, or Updating History Data," in your *Wonderware Historian Administration Guide*.
Manual Data Acquisition Service (MDAS) DLL	Process that can accept non-I/O Server data and send it to the historian to be historized. Data is passed to MDAS through a COM interface. MDAS is used by Wonderware Application Server, the Wonderware Historian OLE DB provider, the event subsystem, and custom client applications.

Component	Description
Manual Data Acquisition Service (MDAS) Server	Windows service that can accept non-I/O Server data and send it to the historian to be historized. Data is passed to MDAS through a data layer. MDAS is used the replication subsystem.
System Driver Service	Internal process that monitors the entire historian system and reports the status with a set of system tags. The system driver also sends data values to the storage subsystem for the current date and time, as well as for pre-defined "heartbeat" tags, such as a discrete system pulse. For more information, see "The System Driver and System Tags" on page 42.

For a complete diagram of the historian architecture, see "Wonderware Historian Subsystems" on page 19.

Data Acquisition from I/O Servers

An I/O Server provides data to Wonderware Historian from the DDE or SuiteLink protocols. I/O Servers accept data values from programmable logic controllers (PLCs), Remote Telemetry Units (RTUs), and similar devices on the factory floor and can forward them on to other software applications, such as InTouch HMI software or the Wonderware Historian.

Note FastDDE is not supported. DDE is not supported if the historian is running on the Windows Server 2003, Windows Server 2008, or Windows Vista operating system.

"Server" and "client" are relative terms. In order for data to be transmitted from the factory floor to a client application on the business LAN, it must be "served up" through a chain of applications. Any application that accepts data from a lower source is a client. Any application that sends data to a higher source is a server. The historian is both a client and a server; it is a client to the I/O Servers and a server to the desktop clients.

To acquire data from an I/O Server, you must add the I/O Server addressing information to the historian database and then associate the I/O Server with an IDAS.

For information on configuring I/O Servers and IDASs, see Chapter 4, "Configuring Data Acquisition," in your *Wonderware Historian Administration Guide*.

I/O Server Addressing

All Wonderware-compatible I/O Servers use DDE addressing, which includes the following distinct parts:

- **Computer name**. This is the node name of the computer runnning I/O Server software.
- **Application name**. This is the name of the application supplying data. The application name can include the name of the computer on which the application is running.
- **Topic name**. A topic is an application-specific subgroup of data elements.
- **Item name**. An item is a data value placeholder.

The format for the addressing is as follows:

\\<*computername*>\<*applicationname*>\<*topicname*>!<*item name*>

The following table provides some examples of DDE addressing.

Address Information	I/O Server	InTouch	Microsoft Excel
application name	\\Computer1\Modbus	\\Computer1\VIEW	\\Computer1\Excel
topic name	ModSlave5	Tag name	Spreadsheet1
item name	Status	ReactLevel	A1 (cell name)

For the Wonderware Historian to acquire data from an I/O Server, the I/O Server addressing information must be added to the overall system configuration. You can use the System Management Console to manually add I/O Server definitions to the system, or you can import I/O Server definitions from existing InTouch applications.

For more information about manually adding I/O Server definitions, see Chapter 4, "Configuring Data Acquisition," in your *Wonderware Historian Administration Guide*.

For more information on importing I/O Server definitions from InTouch HMI software, see Chapter 3, "Importing and Exporting Configuration Information," in your *Wonderware Historian Administration Guide*.

About IDASs

A Wonderware Historian Data Acquisition Service (IDAS) is a small software application that accepts data from one or more I/O Servers or DAServers. An IDAS runs as a Windows service named SysDataAcq. The IDAS processes the acquired data, if necessary, and then sends the data to the Wonderware Historian, where it is subsequently stored.

Note An IDAS was previously called an I/O Driver. IDAS configuration information is stored in the IODriver table in the Runtime database.

When you add an I/O Server definition to the historian, a topic object is created in the associated IDAS. A separate topic object exists for each unique combination of I/O Server computer, application, and topic. Each topic object maintains its own state: idle, connecting, connected, disconnecting, disconnected, overloaded, or receiving. Also, each topic object is assigned a data time-out value based on your assessment of how often data changes for that particular topic.

An IDAS can accept data from one or more I/O Servers, but only sends data to a single historian.

An IDAS can run on the same physical computer as the historian, or on a remote computer. However, only one instance of an IDAS can run on any single computer. Both the IDAS and the historian use NetBIOS network names for communication. Computers must be accessible by those names. Use the `Ping` command to check the availability of the remote IDAS or historian computers.

IDAS seamlessly handles data values, irrespective of their time. For each data point acquired by IDAS, the timestamp, value, and quality are historized in accordance with the storage rules for the tag to which the data value belongs.

For information on configuring an IDAS, see Chapter 4, "Configuring Data Acquisition," in your *Wonderware Historian Administration Guide*.

IDAS Configuration

During normal operation, when the historian is started, it configures an IDAS by sending it information about the tags (including their data sources) for which the IDAS is to acquire data. When the historian storage subsystem is ready to accept data, IDAS automatically connects to its data sources, starts acquiring data, and sends the data to the historian storage subsystem for historization.

The primary purpose for IDAS configuration files is to minimize network traffic and provide information for IDASs configured for autonomous startup. For more information on autonomous startup, see "IDAS Autonomous Startup" on page 81.

The IDAS saves configuration information to a file on the local hard drive in the following folder of the IDAS computer: Document and Settings\All Users\Application Data\ArchestrA\Historian\IDAS\Configurations.

The IDAS configuration file is named as follows:

idatacfg_*SERVERNAME_IDASKEY*.dat

where:

- *SERVERNAME* is the NetBIOS name of the historian computer
- *IDASKEY* is the value of the IODriverKey column for the IDAS in the Runtime database

You can change the IDAS configuration from the System Management Console. The historian dynamically reconfigures itself. If the IDAS is on a remote computer, the historian sends the updated configuration information to the IDAS. The IDAS reconfigures itself and updates the local configuration file. The IDAS continuously acquires and sends data during the reconfiguration process. The historian saves its copy of the updated IDAS configuration file in the following folder of the historian computer: Document and Settings\All Users\Application Data\ArchestrA\Historian\Configuration\IDAS Configurations.

After a successfully configuring IDAS, a copy of the IDAS configuration file is stored on the historian computer. The IDAS configuration file stored on the IDAS computer is identical.

Important IDAS configuration files have a proprietary binary format. Do not modify these files.

If there is more than one autonomous configuration file on the IDAS computer (for example, if you deleted an IDAS on a node while it was disconnected and then added one again), only the newest file is used. A warning is logged on the IDAS computer. For more information on autonomous startup, see "IDAS Autonomous Startup" on page 81.

IDAS Data Processing

An IDAS performs minimal processing on the data values it receives from an I/O Server. The IDAS converts data values into storage data types, depending on the type of the tag associated with the value. For example, if the data value is associated with a floating point analog tag, the incoming value is converted to a floating point value before transmission to the storage subsystem. However, no timestamp conversion is applied because both the I/O Servers and the Wonderware Historian storage subsystem base time on Universal Time Coordinated (UTC).

An IDAS does not apply any storage rules (for example, delta or cyclic storage) unless it is disconnected from the historian and is operating in store-and-forward mode. If the IDAS is operating in store-and-forward mode, all storage rules are applied before the data is stored locally in the store-and-forward history blocks.

Data Transmission to the Storage Subsystem

Data received by the IDAS are stored in a series of 64 KB buffers and are periodically sent to the storage subsystem in packets.

The IDAS sends data from a buffer after the buffer is full, or every second, whichever comes first. The number of pre-allocated buffers is configurable for each IDAS. If you have high data rates for an IDAS and you see log messages indicating the buffers are full, you may need to increase the default buffer count.

For more information, see "Editing Advanced Information for an IDAS" in Chapter 4, "Configuring Data Acquisition," in your *Wonderware Historian Administration Guide*.

IDAS Security and Firewalls

A remote IDAS uses the network account specified at the time of the IDAS installation to communicate with the historian. The historian uses the network account specified at the time of the historian installation to communicate with remote IDASs. You can also change this account after installation using the ArchestrA Change Network Account utility.

To communicate with the historian, the IDAS relies on the security provided by the Microsoft Windows operating system and does not send/receive any user names or passwords over the network.

An IDAS uses named pipes to communicate with the historian and Microsoft file sharing for transmission of store-and-forward history blocks. If a firewall exists between a remote IDAS and the historian computer, the firewall must allow communication using ports from 135 through 139 (TCP/UDP) and port 445 (TCP/UDP).

For more information on IDAS file sharing requirements, see "IDAS Store-and-Forward Capability" on page 79.

IDAS Error Logging

An IDAS logs all errors to the ArchestrA Logger Service. If the IDAS is installed on a remote computer, the ArchestrA Logger Service will also be installed on the remote computer. During normal operation, remote IDAS errors are logged to both the local logger and the logger on the Wonderware Historian computer.

If the network connection between the remote IDAS and the historian fails, no error messages are sent to the logger on the historian computer. Therefore, you should periodically use the System Management Console to check the log on the remote IDAS computer to ensure that no problems occurred. After the network connection is restored, error messages are not forwarded to historian computer.

IDAS Store-and-Forward Capability

IDAS includes "store-and-forward" capability, which protects against a temporary loss of data in the event that a remote IDAS cannot communicate with the Wonderware Historian.

Note The store-and-forward option is not available if you have specified a failover IDAS.

If the remote IDAS cannot communicate with the historian, all data currently being processed can be stored (cached) locally on the computer running IDAS. This hard drive location is called the store-and-forward path and is configurable using the System Management Console.

If the IDAS is unable to send the data to the historian, data is written to this path until the minimum threshold for the cache is reached, at which point no more data is stored. An error message is logged. Remote store-and-forward paths are not supported.

The following actions occur after the historian becomes available again:

- The historian verifies that the IDAS configuration information did not change while the IDAS was disconnected. The historian attempts to restore data transmission from the IDAS. The IDAS stops local data caching and resumes sending data acquired from its data sources to the historian.

- If historian detects a difference between its version of the IDAS configuration, and the IDAS version, it dynamically reconfigures the IDAS to synchronize configuration information. The IDAS applies the changes and updates its local IDAS configuration file. Then, the historian requests restoring data transmission from the IDAS.

- When the IDAS detects availability of the running historian, it sends the store-and-forward data to the historian at the same time it is sending real-time data.

The store-and-forward data is copied to the InSQL8SF$ network share on the historian computer. The data is sent to the historian in chunks, pausing for a configurable amount of time between chunks. The remote IDAS uses the network account specified during the IDAS installation to access the network share.

If the data stored on the remote IDAS node does not constitute a complete history block (for example, due to a power failure), the data is still forwarded to the historian and historized. A message is logged that an incomplete block was processed.

If an error occurs when a store-and-forward block is processed, the block is moved to the \Circular\Support directory. A message is logged. Blocks are not automatically deleted from the \Support directory. You must manually delete blocks to prevent them from consuming disk space allocation for the circular storage location. If you upgraded from the Wonderware Historian 8.0.x, then the block is moved to the existing \Log directory.

After data from the store-and-forward cache is sent to the historian, the cache is deleted from the IDAS computer.

Enabling IDAS store-and-forward mode increases system resources used by the IDAS service because the store-and-forward subsystem must be initialized and then maintained in standby mode, ready to accept data.

If the historian computer has sufficient system resources, you can configure the local IDAS to continue store-and-forward data collection even if the storage subsystem is stopped.

IDAS Redundancy

For each IDAS that you define for the system, you can specify a "failover" IDAS. If the Wonderware Historian stops receiving data from the primary IDAS, it automatically switches to the failover IDAS. The switch may take some short period of time, and some data may be lost during the transition.

Note You cannot specify a failover IDAS for an IDAS that has store-and-forward functionality enabled. These two features are mutually exclusive. Applications that require both failover and store-and-forward functionality should use a redundant Wonderware Application Server with RedundantDIObjects.

IDAS Autonomous Startup

Normally, the Wonderware Historian Configuration Service starts an IDAS. However, a remote IDAS that is enabled for store-and-forward can be configured to start independently of the historian. Autonomous startup is useful when the historian is unavailable due to a network failure or when the historian is not running when the remote IDAS computer starts. Using autonomous startup, the IDAS starts caching store-and-forward data without waiting for a command from the historian.

For an IDAS to autonomously start, it must be configured to acquire data from at least one data source. During the configuration process, the IDAS must be connected to the historian to ensure that the configuration file is created on the local IDAS computer. An autonomous startup requires an existing local IDAS configuration file on the IDAS computer, so that it has all of the information it needs to begin acquiring data. For more information, see "IDAS Configuration" on page 76.

An IDAS can start autonomously by being started manually from the Windows Services console or by starting automatically after a computer restart.

When an IDAS starts, it attempts to load the configuration information from the local configuration file. If it is able to do so, the IDAS uses that information to connect to its data sources and start acquiring data. As soon as the internal IDAS data buffers are full, the IDAS switches to store-and-forward mode and stores data to the local hard drive.

If there is more than one autonomous configuration file on the IDAS computer (for example, if you deleted an IDAS on a node while it was disconnected and then added one again), only the newest file is used. A warning is logged on the IDAS computer.

If the local configuration information cannot be loaded, the IDAS remains in an idle state until contacted by the historian. If the IDAS is not contacted by the historian within the default start time-out of 60 seconds, the IDAS shuts down. Note that the IDAS startup time-out is different than the time-out used by the IDAS during autonomous startup. Information on changing the default IDAS startup time-out is provided in a TechNote, which is available from technical support.

When the historian becomes available, data transmission from the IDAS will be restored. For more information, see "IDAS Store-and-Forward Capability" on page 79.

Even if the IDAS is configured for autonomous startup, under certain circumstances it may be started by the Wonderware Historian Configuration service. The autonomous startup time-out is the time, in seconds, that an autonomous IDAS waits for configuration commands when started by the Configuration service before switching to autonomous mode. The autonomous startup time-out does not apply when you start the IDAS either manually using the Windows Services console or by configuring the IDAS to automatically start (autostart) using the Windows Services console.

If an IDAS is configured as autonomous, the startup type for the SysDataAcq service are changed to Automatic, and the IDAS starts every time the IDAS computer is restarted. If the IDAS is then changed to be non-autonomous, the startup type will be changed back to Manual. Information on changing this default behavior is provided in a TechNote, which is available from technical support.

IDAS Late Data Handling

"Late" data arrives at the Wonderware Historian with a timestamp later than 30 seconds of the current historian time. The storage subsystem can cater for processing different types of late data from an I/O Server. For example:

- Data that is late, but is sent in steady stream. For example, because of communications delays, a topic defined in an I/O Server might send a stream of data values that is consistently two to three minutes behind the Wonderware Historian time. Although this type of data is late, it is handled by the real-time data storage process if the topic is configured for late data.

- Data that is sent in periodic bursts. For example, a topic defined in an I/O Server might represent an RTU that sends a block of data values every few hours. Late data of this type is handled by the "manual" data storage process.
- Late data must not encompass more than six history blocks at any given point of time."

The late date option must be enabled for the topic in order for late data to be processed. If the late data option is not enabled, any data values from that topic later than 30 seconds behind the Wonderware Historian are discarded. Any discarded data is reflected in the SysPerfDataAcq*N*OutsideRealtime system tag for the IDAS.

For more information on enabling late data, see "Editing Advanced Information for an IDAS" in Chapter 4, "Configuring Data Acquisition," in your *Wonderware Historian Administration Guide.*

Important Support for late data is not intended to accommodate time synchronization problems between the IDAS, I/O Servers, and the historian. It is imperative that you properly synchronize the IDAS, I/O Servers, and historian time. If the IDAS is sending data in a steady stream outside of the real-time window, it is likely there is a time synchronization problem. For more information, see "Time Synchronization for Data Acquisition" on page 87.

Regarding data throughput, the following rules apply:

- If the late data option has been enabled for a topic and the data is within the real-time window, the system will support a late data throughput equal to its real-time throughput capability.
- If the late data option has been enabled for a topic and the data is outside the real-time window, the system will support a late data throughput of one percent of its real-time throughput capability.

If you enable late data for a topic, you need to configure the following two parameters:

- **Idle duration**. The idle duration is a delay in processing the data from the topic. For example, an idle delay of 60 seconds means that data from the topic is cached and only processed by the historian storage subsystem after no more data has been received from the topic for at least 60 seconds. By default, the idle duration is set to 60.

  The idle duration is important if you anticipate bursts of late data being sent to the IDAS. The setting you choose depends on your specific historian implementation and application requirements, because the higher you set the idle duration, the longer it will take to see the data, for example, in a historian client application. Also, if you are trending blocks of late data, there will be no gap (NULL values) to indicate the end of the data burst. Instead, the trend pen will "flat line" until the next block of late data starts.
- **Processing interval**. The processing interval is a safety precaution. In case the nature of the data is such that the idle duration is never satisfied, the historian storage subsystem will process data from the topic at least once every processing interval. By default, the processing interval is set to twice the idle duration. The processing interval cannot be set to a value less than the idle duration.

The processing interval is important if you anticipate a steady stream of late data being sent from the IDAS. The higher you set the processing interval, more memory will be used by the historian, but no more than 32 MB in total. When that limit is reached, all accumulated data is processed.

Whenever possible, it is better to set up a remote IDAS on the I/O Server computer to handle the data processing, instead of using the late data settings for the topic.

If you are using a remote IDAS with several late data topics, the lowest settings for any topics configured for the IDAS will apply.

All data values received as late data from an autonomous IDAS are stored in delta mode, regardless of whether a tag is configured for cyclic storage. Therefore, you should configure the tag to be stored in delta mode, if you expect the data to be late. Also, if late data is enabled for a topic, the timestamp for the initial value will never be overwritten by the storage subsystem, no matter how early it is. This differs from how real-time data is handled. For real-time data, if the storage subsystem receives an initial data value that has a timestamp that is older that the system start time, it will change the timestamp of the first value to be the current time of the historian.

If you enable late data for an autonomous IDAS, the topic time-out will automatically be set to 0 and disabled.

Important If the topic is configured to receive late data, the storage subsystem never receives disconnect/reconnect values from the IDAS for tags belonging to that topic. You may see "flat line" on a trend even if the data source is disconnected from the historian.

Support for Slow and Intermittent Networks

You can configure an IDAS to accommodate data transfer over a slow and/or intermittent network. A network is considered intermittent if it fails for short periods (a few seconds at a time) at random intervals. A slow network can have transfer rates as low as 56Kb/s or periods of large data transfers that use most of the available bandwidth.

For an intermittent network, you may need to set the minimum store-and-forward duration to a value higher than the default. This prevents the IDAS from frequently going in and out of store-and-forward mode due to the brief periods of network unavailability. Typically, the default duration is appropriate in all but the worst cases.

For a slow network, you can make the following adjustments:

- Set the file chunk size to a smaller value. The file chunk is the block of data that is sent from the IDAS to the Wonderware Historian at one time. Sending smaller chunks reduces the network load at any given time. A good indication that the file chunk size is too big is that when the forwarding operation starts, the IDAS immediately goes back into store-and-forward mode; the forwarded chunk is large enough to overload the network, causing the IDAS to detect a problem and switch into store-and-forward mode.
- Increase the forwarding delay. The delay is the interval at which the IDAS sends chunks of the store-and-forward data to the historian. Increasing the delay further spreads out the network load for the forwarding operation.
- Increase the autonomous start time-out for IDASs configured to autonomously start. An indicator that the time-out period is too small is that when the remote IDAS starts by the Wonderware Historian Configuration service, it goes into store-and-forward mode, and then later restores the connection to the historian.
- Increase the connection time-out. One symptom that the time-out is too small is that the network connection is physically fine, but you see error messages in the log regarding IDAS time-out problems.

A remote IDAS can sustain a network interruption as long as there is disk space available to hold the local store-and-forward data.

I/O Server Redundancy

You can edit an I/O Server definition to include a "failover" I/O Server. This alternate I/O Server can be installed on the same computer as the primary IDAS or on another computer. If the network connection between the primary I/O Server and the IDAS fails, the IDAS automatically switches to the alternate I/O Server, provided that the alternate I/O Server is running. The switch may take some short period of time, and some data points may be lost during the transition.

Redirecting I/O Servers to InTouch HMI Software

When you redirect an I/O Server to InTouch HMI software, you are specifying to acquire tag values from a particular InTouch node that is using an I/O Server, instead of acquiring them directly from the I/O Server. This feature is useful when you need to reduce the loading on the I/O Server, or if the InTouch node is more accessible on the network.

When you redirect the I/O Server, the computer name and I/O Server type will reflect the InTouch node as the I/O Server from which data is acquired. For example, suppose you were using the a Modicon Modbus I/O Server on computer "I23238." The application name for the I/O Server address appeared as **\\I23238\modbus**. If you redirect this I/O Server to the InTouch node "InTouchNode1," then the address will be modified to reflect \\InTouch**Node1\view**.

Time Synchronization for Data Acquisition

All I/O Servers that support SuiteLink timestamp plant data as it is acquired. It is important to understand how synchronization is handled between the timestamps for I/O Server, the computer clock for the IDAS(s), and the computer clock for the Wonderware Historian(s). An overview of time synchronization is as follows:

1. If you have multiple historians on your network, you should synchronize all computer clocks to a single historian designated as a master time server. You could then synchronize this master historian to an external time source. Designate the master historian using the *TimeSyncMaster* system parameter.
2. Periodically, a historian automatically synchronizes the computer clock of any remote IDASs to its own computer clock. The IDAS synchronization is enabled by means of the *TimeSyncIODrivers* system parameter.

3. Every hour, an IDAS automatically synchronizes the *timestamping mechanism* of any associated I/O Servers with its own computer clock. This does not actually change the system clocks of any I/O Server computers. Instead, the difference in the system clocks on the two computers (I/O server and Historian) are determined, and a bias is calculated that is then applied to all values from that I/O server computer. For example, if the historian clock is seven seconds ahead of the I/O Server computer's clock, SuiteLink adds seven seconds to every timestamp from the I/O Server. If a topic is disconnected/reconnected due to a topic time-out or other communications failure, the I/O Server timestamping is not updated until the time synchronization interval has passed. You can change the frequency of the synchronization using the *SuiteLinkTimeSyncInterval* system parameter.

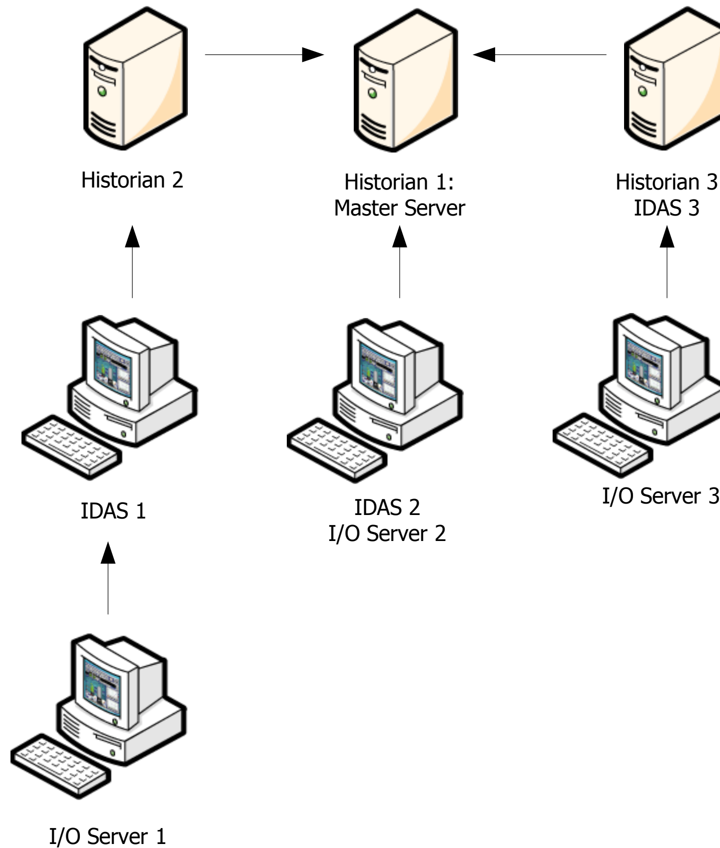
   The SuiteLink protocol also does some time adjustments to keep timestamps consistent across nodes. SuiteLink bases this adjustment on the time difference detected at startup and each hour. For example, NodeA and NodeB have a time difference of 17 seconds. The I/O Server is on NodeA, and the IDAS is on NodeB (either local to the historian or a remote IDAS for a historian on another NodeC). When the I/O Server on NodeA timestamps a value at 12:00:00.000, it is transmitted to NodeB with an adjusted timestamp of 12:00:17.000. If the historian is configured to timestamp at the source, this value is stored with a timestamp of 12:00:17.000. If, instead, the historian is configured to timestamp at the server, and there is a two-second communications latency, then the value is stored with a timestamp of 12:00:19.000.

   For normal operations on systems with synchronized clocks, there is no adjustment made by SuiteLink and everything operates as expected. However, when either the systems are out of synch, or even were out of synch when SuiteLink communications between the nodes started, the timestamps will be adjusted. Because of the way SuiteLink adjusts timestamps, it is easy to produce misleading results if system tests involve adjusting system clocks on the systems, because SuiteLink does not immediately update its time skew.

Note Time synchronization does not apply to I/O Servers that use DDE because these servers do not perform timestamping. The time of the IDAS computer is always used for data coming from DDE from I/O Servers.

For more information on setting system parameters, see "Editing System Parameters" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide*.

The following diagram shows an example of how computers can be synchronized to a single time:



For an MDAS-enabled client application, you can use the `net time` command (for the Windows operating system) to synchronize the client computer's clock to your master historian.

A good indicator that a time synchronization problem is occurring is if the value of the SysDataAcq*N*OutsideRealtime system tag is high. This tag indicates how many values received from a particular IDAS are outside the limits of the real-time window. For more information on system tags, see "The System Driver and System Tags" on page 42. For more information on the real-time window, see "About the Real-Time Data Window" on page 97.

Data Acquisition by Means of INSERT and UPDATE Statements

You can insert or update history data in the Wonderware Historian extension tables using Transact-SQL INSERT and UPDATE statements.

For more information, see Chapter 6, "Importing, Inserting, or Updating History Data," in your *Wonderware Historian Administration Guide*.

Data Acquisition from MDAS

The manual data aquisition service (MDAS) accepts data (real-time data, as well as inserts and updates) from its host and forwards it to the Wonderware Historian storage subsystem. For example, Wonderware Application Server uses MDAS to send history data to the historian. Replicated data from a tier-1 historian is sent to the historian using MDAS.

The storage subsystem merges data acquired from MDAS to existing historized data. All data can be accessed from the History, WideHistory, and Live extension tables.

MDAS is implemented in two ways within the system: one as a client-side Windows DLL and one as a Windows service.

The DLL version of MDAS uses DCOM and file shares to send data to the historian. For both the MDAS and Wonderware Historian computers, make sure that DCOM is enabled (not blocked) and that TCP/UDP port 135 is accessible. The port may not be accessible if DCOM is disabled on either of the computers or if there is a firewall between the two computers that blocks the port. For information on enabling DCOM communication through a firewall, see your Microsoft Windows operating system documentation.

The Windows service version of MDAS is used only for the replication subsystem. The MDAS service manages data transmission between tier-1 and tier-2 servers. The Windows service version of MDAS communicates using TCP-based communication layer through an open port that you specify when you set up the replication server. The MDAS service also handles security-related error reporting on the tier-2 historian.

Note Data acquired through the manual data acquisition service is NOT stored in the ManualAnalogHistory, ManualDiscreteHistory, or ManualStringHistory tables. Prior to Wonderware Historian version 8.0, these tables were the only mechanism to store manually acquired data. Currently, these tables are provided for backward compatibility only.

Importing Data from a CSV File

Legacy data can be batch imported into the Wonderware Historian from a CSV file. A batch import allows you to import legacy data from other historians. If you want to import history data from an InTouch application, you can use the InTouch History Importer, which converts InTouch history files to the required CSV format. Imported data is integrated with data currently stored in history blocks, providing you with seamless access to all your data.

For more information, see Chapter 6, "Importing, Inserting, or Updating History Data," in your *Wonderware Historian Administration Guide.*

Chapter 5

Data Storage Subsystem

The Wonderware Historian storage subsystem saves plant data from various sources to disk. The storage subsystem stores data for analog, analog summary, discrete, state summary, string, and system tags in sets of files on disk called history blocks.

Historical data can be retrieved by sending SQL queries through the Wonderware Historian OLE DB provider, which is part of the data retrieval subsystem. At retrieval, the historized tag data is presented as if it resided in SQL Server tables. For more information, see Chapter 6, "Data Retrieval Subsystem."

The storage subsystem processes only historical data; it does not handle configuration data. For more information about saving configuration data, see Chapter 3, "Configuration Subsystem."

Also, the storage subsystem does not handle event data, including summarized data created by the event subsystem. For more information about event data, see Chapter 10, "Event Subsystem."

You use the System Management Console to configure all aspects of the storage subsystem. For more information, see Chapter 5, "Managing Data Storage," in your *Wonderware Historian Administration Guide*.

Storage Subsystem Components

The components of the storage subsystem are:

Component	Description
Realtime Data Storage Service (aahStoreSvc.exe)	Internal process that stores real-time data to disk. This process runs as a Windows service.
Manual Data Storage Service (aahManStSvc.exe)	Internal process that processes non-real-time data and stores it to disk. This process runs as a Windows service. This process is also called "alternate" storage.
Active Image	Memory segment that temporarily hold all real-time data while the storage subsystem stores the actual values to disk.
History Blocks	Set of folders and files on disk that contain historical data.
History Cache	Allocation of memory in which history block information is loaded to increase data retrieval efficiency.
Tier-2 Storage Process (aahStorageEngine.exe)	Secondary storage process that handles replication data on a tier-2 historian. This is not a Windows service.

For a complete diagram of the Wonderware Historian architecture, see "Wonderware Historian Subsystems" on page 19.

Storage Data Categories

Data saved to the Wonderware Historian belongs in the following categories: real-time data, "late" data, and "old" data. Each type of data has a separate set of characteristics and is handled differently by the historian. These characteristics are:

- Time sequential data flow. The data acquired by the historian can be in either time sequential order or in any order. For time sequential data, each consecutive data value received has a timestamp that is later than the previously received value. Data coming from an I/O Server is typically time sequential. Blocks of data that are imported into the system do not necessarily follow each other in time and would be an example of non-time sequential data.

- Relationship to the system-wide real-time data "window." The real-time window is the maximum delay, relative to current time of the server, for which data is considered real-time by the system. For more information, see "About the Real-Time Data Window" on page 97.
- Support for "future" data. If the incoming data has a timestamp that is in the future, relative to the server time, it will be handled differently based on what data category it falls in.

	Real-Time Data	Late Data	Old Data
Data Flow	Timestamps must be in time sequential order.	Timestamps must be in time sequential order.	Timestamps can be in any order.
Real-time Window Application	The timestamp for the data value must fall within the time window.	The timestamp for the data value can fall either inside or outside of the time window, depending on whether the late date setting is enabled for the topic.	The data can have any timestamp.
Support for Future Data	Yes. If a value with a future timestamp is later overwritten by another data value, the QualityDetail value will reflect the overwrite.	N/A. Late data always has a timestamp in the past, relative to the server time. Timestamps for consecutive late data values can shift forward and backward in time and still be considered late. If the timestamp for a received late data value is earlier than the last stored value, then the value with the later timestamp is overwritten, and the QualityDetail value reflects the overwrite.	No. Old data with a timestamp in the future is discarded.

	Real-Time Data	Late Data	Old Data
Typical Sources of Values	• I/O Servers • System tags • Transact-SQL INSERT statements where wwVersion = REALTIME • Data from Wonderware Application Server • Tier-1 historian • Historian SDK	• I/O Servers (RTUs) • "Fast load" .CSV imports. • Tier-1 historian • Historian SDK	• Transact-SQL statements • Regular .CSV imports • Tier-1 historian • Historian SDK
Storage Service	Data is processed by the Realtime Data Storage Service (aahStoreSvc.exe).	Data can be processed by either type of storage, depending on whether or not the timestamps fall within the block of data that is currently being processed (the current snapshot). Data is always processed by Realtime Data Storage Service (aahStoreSvc.exe) first.	Data is processed by the Manual Data Storage Service (aahManStSvc.exe)

If necessary, incoming data timestamps are converted to Universal Time Coordinated (UTC) before storing the data.

The Realtime Data Storage Service (aahStoreSvc.exe) and the Manual Data Storage Service (aahManStSvc.exe) work together to store all of data to disk, organize the data in such a way that, upon retrieval, the data is as seamless and integrated as possible.

About the Real-Time Data Window

The real-time "window" is the maximum delay, relative to current time of the server, in which data is considered real-time by the system. The real-time window can range from -30 seconds to +999 milliseconds of the current server time.

The following rules apply when storing data with timestamps relative to the real-time window:

- If the late data setting for a tag topic is not enabled and the received data is more than 30 seconds late, the value is discarded, and a warning is logged. If the received data value is within 30 seconds of the server time, it is stored as received.
- If the late data setting for a tag topic is enabled and the received data value is within the real-time window, then the value is stored by the real-time storage service with no changes. If the received data value is outside of the real-time window, then the value is passed to the alternate storage services and stored without changes.
- If the late data setting for a tag topic is enabled, the received data value is stored in delta mode, even if the tag is configured for cyclic storage and the received data value is within the real-time window.

For more information on the late data setting for a topic, see "IDAS Late Data Handling" on page 82.

You can adjust the real-time window for “late” data topics by configuring the RealTimeWindow system parameter. The real-time window for regular real-time data (not "late") is fixed as 30 seconds. For more information, see "Editing System Parameters" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide*.

You must balance the need for a larger real-time window with the amount of memory available on your computer. If your system has a large tag count or high data throughput, increasing the real-time window increases the memory requirements for storage because the storage system must process more data as real-time data, which is more resource-intensive than the storage of late or old data.

Adjusting the real-time window also has implications if you are using delta storage with a swinging door deadband. For more information, see ""Swinging Door" Deadband for Delta Storage" on page 101.

If the system does not have enough memory to adequately process real-time data, the window is adjusted internally. An appropriate message is logged. The value of the RealTimeWindow system parameter, however, remains unchanged.

Important The real-time window is not intended to accommodate time synchronization problems between an IDAS and the historian. It is imperative that you properly synchronize the IDAS and historian. If the IDAS is sending data in a steady stream outside of the real-time window, it is likely there is a time synchronization problem. For more information, see "Time Synchronization for Data Acquisition" on page 87.

If a data value is discarded because it did not fit the requirements of the real-time window, the historian logs a warning message. Warning messages are logged at one intervals during the period when data is being discarded.

Data Modification and Versioning

You can use the Wonderware Historian to modify historized data by either inserting one or more new values into history after the original data for that time period had been historized, or by updating a value or a time region of values. All modifications are versioned, and previous versions are preserved in all cases, allowing you to view the original data before the modifications.

The term "original data" refers to the original set of values historized for a tag. For example, a real-time stream of data from an I/O Server represents original data. Inserts or updates performed on the original stream of data results in a new version. Data modification can be performed on any tag category (I/O Server tags, system tags, or manual tags). For manual and I/O Server tags, new data can be presented as either original data (non-versioned) or inserted data (versioned).

The historian supports the viewing of only the original version or the latest version of data. Interim versions are preserved in history but are not exposed through the retrieval layer. The QualityDetail column contains a special, unique value to indicate that the data represents a "modified" value. That is, the current value is the latest version of the data, but an earlier version of the data is present in history.

Storage Modes

When you define a tag, you need to specify a storage method, which specifies how the tag's data is saved as historical records. For example, the system may be receiving 100 values per millisecond for an I/O Server tag, but you might not want to store all of these values. How you configure the storage method affects the number of values written to disk and the resolution of the data that will be available later for retrieval.

The following types of storage modes are available:

- No data values are stored.
- All data values are stored (forced storage).
- Only changed data values are stored (delta storage).
- Only data values that occur at a specified time interval are stored (cyclic storage).

"Forced" Storage

Forced storage is the storage of each value as it comes in from the plant floor or from a client application, without filtering applied (that is, no delta or cyclic storage mode is used).

Forced storage is useful, for example, if you have an I/O Server that collects data by exception from the plant floor device (instead of using a polling interval). You can allow the I/O Server to filter the values by exception, instead of the Wonderware Historian.

If the I/O Server uses a polling interval, then storage with no filtering is similiar to cyclic storage. However, the cyclic storage rate determines the values that are stored, and that rate can only go down to a one second resolution. If you use storage with no filtering, the I/O Server controls which values are stored, and these can be at a finer resolution than one second. For forced storage, if the data source is sending the same values at each internal scan cycle, then each value is stored.

Delta Storage

The Delta storage mode stores data based on a change in a value. Delta storage writes a historical record only if the current value changes from the previous value. Delta storage is also called "storage by exception." Delta storage is typically used to store discrete values, string values, and analog values that remain constant for long periods of time. For example, you don't want to store the value of the discrete tag "PUMPON" every ten seconds if the pump usually stays on for months at a time. The value is stored along with a timestamp of when the change occurred, to an accuracy of 1 ms.

The following types of deadbands can be applied for delta storage:

- Time deadband
- Value deadband
- Rate of change (swinging door) deadband

Note You can retrieve down to the millisecond by using a CONVERT clause in an OPENQUERY statement with specific style specifiers of the Wonderware Historian. For more information, see "Querying Data to a Millisecond Resolution using SQL Server 2005" on page 310.

Time and Value Deadbands for Delta Storage

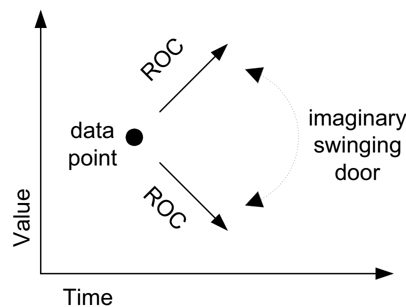
To further decrease the resolution of tag values stored in delta mode, use a time deadband or a value deadband.

- A time deadband is the minimum time, in milliseconds, between stored values for a single tag. Any value changes that occur within the time deadband are not stored. The time deadband applies to delta storage only. A time deadband of 0 indicates that the system will store the value of the tag each time it changes.
- A value deadband is the percentage of the difference between the minimum and maximum engineering units for the tag. Any data values that change less than the specified deadband are not stored. The value deadband applies to delta storage only. A value of 0 indicates that a value deadband will not be applied.

"Swinging Door" Deadband for Delta Storage

A "swinging door" deadband is the percentage of deviation in the full-scale value range for an analog tag. The swinging door (rate) deadband applies to delta storage only. Time and/or value deadbands can be used in addition to the swinging door deadband. Any value greater than 0 can be used for the deadband. A value of 0 indicates that a swinging door deadband will not be applied.

The swinging door deadband is essentially a rate of change deadband, based on changes in the slope of the incoming data values. For example, specifying a swinging door deadband value of 10 percent means that points will be stored if the percentage change in slope of the consecutive data values exceeds 10 percent. The percentage of allowable "swing" in the data values gives this type of deadband its name.



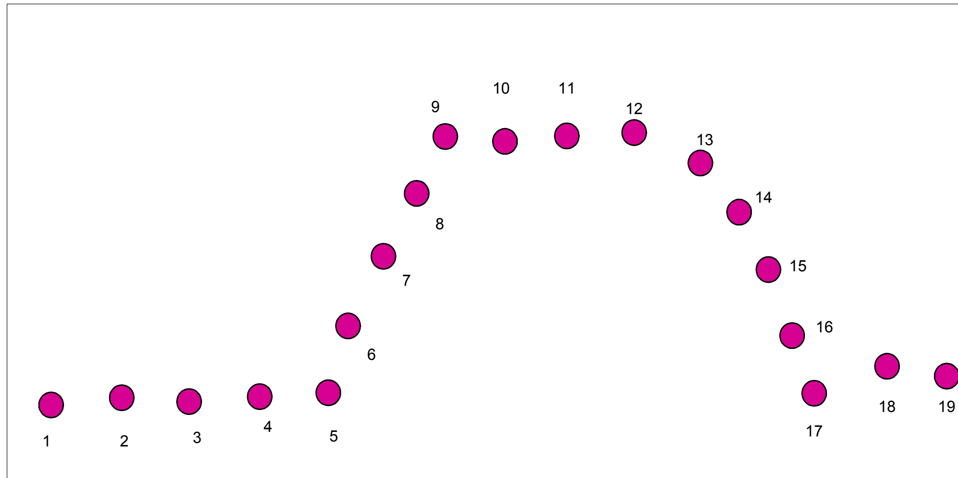
Benefits of the Swinging Door Deadband

One benefit of using a swinging door deadband is that it reduces the disk space required to store data. Another benefit of the swinging door deadband is that it captures the data value before the rate change, which is something that a value deadband does not do. If you trend data, the peaks and valleys of the trend curve are more defined to provide a more accurate picture of what is happening in your plant.

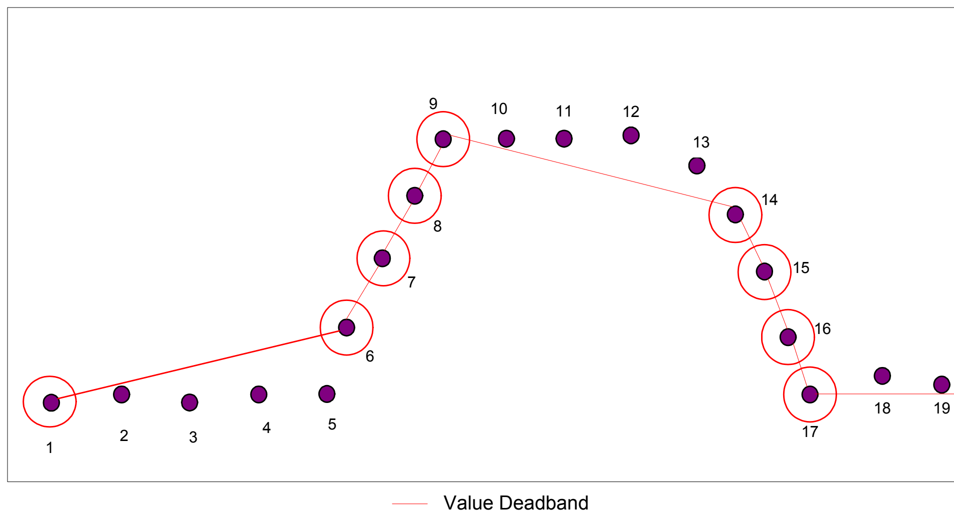
Generally, using a swinging door (rate) deadband provides better representation of the value change curve with the same or less number of values stored than regular value or time deadbands for delta storage.

The following graphics compare the trend curves of the same raw data, but with different deadbands applied.

The following graph shows the trend of the actual raw data values:

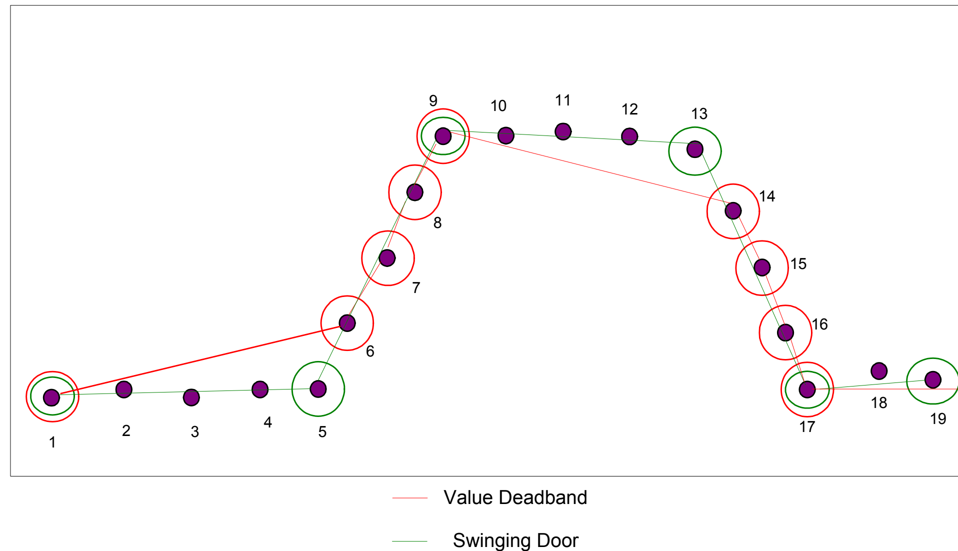


The following graph shows the trend of the data values with a value deadband applied. Notice how only the first data value that deviates by the deadband from the previous value will be stored, and not any of the values between the starting value and the first deviating value.



Value Deadband

The following graph shows the data values that will be stored for both a value deadband and a swinging door deadband. Notice how the swinging door deadband captures data before the deadband change, allowing for a more complete view of the data.



A swinging door deadband is most useful for tags that have a steady increase and decrease in slope, such as a tank level or tank temperature that rises and falls. A swinging door deadband may not be appropriate for "noisy" signals, in which the value of the tag constantly fluctuates around a certain point for long periods of time. Also, the reduction in storage requirements offered by the swinging door deadband may not have much of an impact if you have an application with a small tag count (for example, 500 tags). In this case, it may not be necessary to use a deadband at all.

A swinging door deadband is applicable for analog tags that receive data from the following sources:

- Real-time data values from I/O Servers or MDAS
- Store-and-forward data from a remote IDAS
- Late data from an I/O Server topic that was configured for late data
- A "fast load" .CSV import
- Real-time inserts of data using a Transact-SQL statement

A swinging door deadband is not applicable for manual inserts of data through a .CSV import of a Transact-SQL statement.

To best visualize the tag that uses swinging door storage, plot a trend using the Historian Client Trend application and set the plot type from to "line" (rather than "step-line").

Additional Options that Affect the Swinging Door Deadband

The swinging door deadband (the rate deadband) can optionally be combined with a value deadband and/or a deadband override period, the combination of which will affect which values are actually stored. The behavior of the swinging door algorithm also depends on the value of the real-time window in the Wonderware Historian, as specified by the RealTimeWindow system parameter.

- Value deadband

  When combined with rate deadband (with or without a deadband override period), the value deadband is always applied first, followed by the other deadbands. For the value deadband, the system checks the difference in value between the received point from the value of the last stored point. Only when this difference exceeds the value deadband does the system consider the point for rate evaluation.

- Deadband "override" period

  If the elapsed time since the last stored point exceeds the deadband override period, the last received point before the time at which the deadband override period expired is stored, regardless of value and rate deadband.

- Real-time window

  The real-time window setting (RealTimeWindow system parameter) allows for the expansion of the time window for which the storage system considers data to be "real-time." The real-time window is important for swinging door deadbanding because it determines the maximum length of time that a point will be "held" by the storage system without storing it, while waiting for the next point. For more information, see "About the Real-Time Data Window" on page 97.

  Real-time window and deadband override periods are two independent modifiers that force the storage of received points that may have otherwise been discarded due to the setting of either the rate deadband or the value deadband.

- The real-time window specification is more likely to select points for storage when the time period between points received from the source is less than the real-time window, but the slope of the incoming data values is such that the rate deadband excludes the points from being stored.
- The deadband override period is more likely to select points for storage if the rate at which points are received from the data source is slow (slower than the real-time window) and the rate deadband excludes the points from being stored.

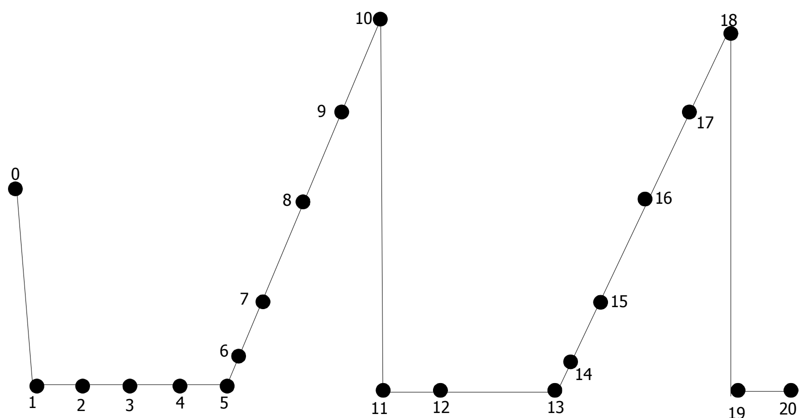
For an illustration of how these factors work together to determine the actual values to be stored, see "Swinging Door Deadband Example" on page 105.

Whatever the combination of rate deadband, value deadband, and deadband override period specified, only points actually received from the data source are stored on disk. That is, points to be stored on disk are never "manufactured" by the swinging door algorithm. This is particularly relevant in understanding the behavior implied by specifying the real-time window and the deadband override period.

Swinging Door Deadband Example

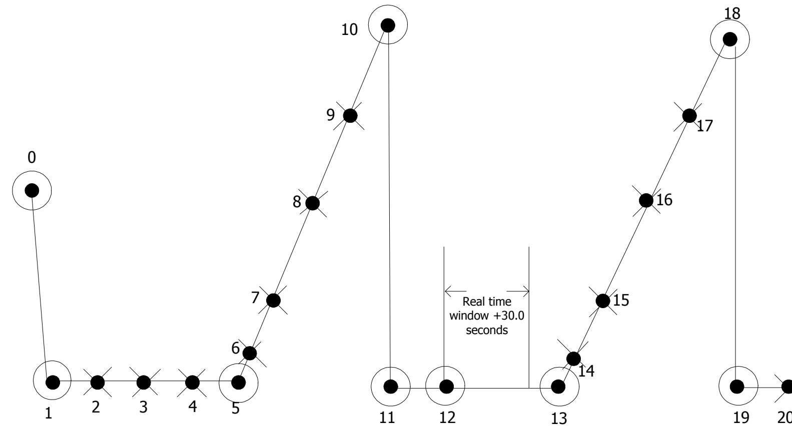
The following examples illustrate the effects of the different swinging door options.

All of the examples are based on the following raw data. The numbered points represent actual values received from a data source.



Swinging Door Deadband: Rate Only

The following diagram depicts an ideal case, where the incoming signal is noise-free and with a proper rate deadband specification only (no value deadband or deadband override period).



Assume point 0 has been stored on disk. The system waits for point 2 to arrive before making its next storage decision. When point 2 is received, the storage engine calculates the change in slope as follows:

Slope0_1 is considered the base slope, and Slope1_2 is considered the current slope.

Slope0_1 = (Value1 - Value0) / (Time1 - Time0)

Slope1_2 = (Value2 - Value1) / (Time2 - Time1)

Slope_Change_Percent = 100* | (Slope1_2 - Slope0_1) / Slope0_1 |

If

Slope_Change_Percent > Rate_Deadband_Specified

In other words, if the percentage change in slope is greater than the specified rate deadband, the storage engine goes ahead and stores point 1 on disk. Next, it receives point 3. The base slope for point 2 will be the slope between points 1 and 2. The current slope will be the slope between points 2 and 3 only if point 1 was stored. If point 1 was not stored, then the base slope for point 2 will be the slope between points 0 and 1, and the current slope will be the slope between points 2 and 3.

The base slope for an evaluation point is not changed unless the previous point is stored; otherwise, the base slope will be the last known current slope that caused a point to be stored on disk.

Assuming point 1 is stored, because the slope between points 2 and 3 is about the same as the slope between points 1 and 2, the rate deadband criterion is not satisfied, and point 2 is discarded. When point 4 is received, the slope change calculation results in point 3 being discarded, and so on until point 6 arrives. Now the rate deadband criterion is satisfied (slope change between points 5 and 6 and points 1 and 2 is greater than the rate deadband specified), and point 5 is stored on disk.

The arrival of point 7, likewise, discards point 6 even though the actual slope between point 6 and point 7 may be quite high, and may even be higher than the rate deadband specified, it is not sufficiently different from the slope between points 5 and 6 to qualify point 6 to be stored. Following this logic through until point 12 is received results in the storage on disk of points 10 and 11, discarding all the other points in between.

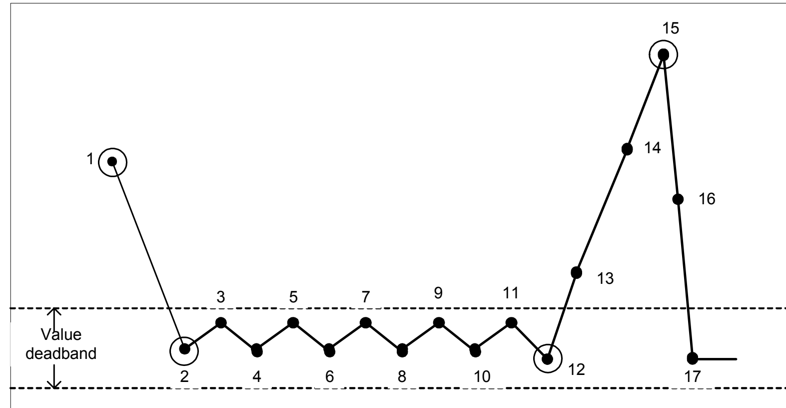
Point 13 illustrates the effect of the real-time window setting. Under normal circumstances, point 12 would not qualify to be stored. If, however, the elapsed time between receiving point 12 and point 13 exceeds the time window in which the storage engine is able to store point 12 as a real-time point, point 12 is stored anyway, and the value of the SysRateDeadbandForcedValues system tag is incremented. In other words, if, while the system waits for point 13 to arrive, the timestamp of point 12 becomes so old that it reaches the limit for the real-time window, point 12 is stored regardless of whether it is outside the deadband.

The SysRateDeadbandForcedValues system tag counts the number of "extra" points stored as a result of an insufficient real-time window for swinging door storage.

When point 14 arrives, the base slope for evaluating point 13 is between points 11 and 12, and not between points 12 and 13, because point 12 was stored due to the real-time window expiration. A point stored due to the real-time window does not re-establish the base slope; only points stored due to exceeding the rate change causes the base slope to be re-established. Then "normal" rate change evaluation resumes, resulting in point 13 being stored, and so on.

Swinging Door Deadband: Rate and Value

In the following diagram, a signal with some "noise" is shown. The effect of applying both a rate and value deadband to swinging door storage is illustrated. The value deadband is indicated by two horizontal dashed lines.

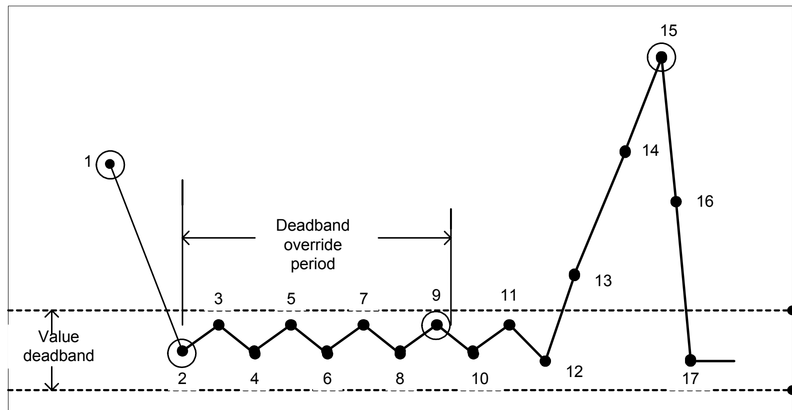


Assume that point 1 has been stored to disk. Point 3 passes the value deadband check, allowing points 2 and 3 to be evaluated for rate change. Assuming that the point exceeds the rate change requirement, then point 2 is stored. Until point 13 is received, all intermediate points are discarded by the value deadband filter. In this example, it is assumed that the change in slope between points 2 through 3 and points 12 through 13 is greater than the rate deadband, so point 12 is stored on disk. When point 14 is received, the normal operation begins.

If a rate deadband is applied without a value deadband, all of the "noisy" points (3 through 11) would have been stored, because the slope of the signal changes radically between successive points. The value deadband removes the noise, but also introduces some amount of distortion in the resultant signal.

Swinging Door Deadband: Rate, Value, and Deadband Override Period

The following graphic illustrates the effect of a rate deadband combined with a value deadband and a deadband override period.



Assume point 1 is stored to disk. Point 3 makes it through the value deadband check, allowing points 2 and 3 to be evaluated for rate change. Assuming the point exceeds the rate change requirement, then point 2 is stored.

Adding a value deadband alone could result in distortion of the stored data.

For example, suppose that the rate deadband is specified such that point 12 does not get stored. That is, the change in slope between points 2 through 3 and points 12 through 13 is not greater than the rate deadband. In that case, the data representation (points 1, 2, and 15) is grossly distorted because the value deadband is discarding key points.

To allow for better representation, a deadband override period may optionally be specified. If the elapsed time between the last stored point and the currently received point is more than the specified deadband, then the point immediately prior to the currently received point is stored. In this example, the elapsed time between point 2 and point 10 is more than the deadband, so point 9 is stored. The data actually stored to disk (points 1, 2, 9, and 15) is a better approximation of the original data.

It is important to note that after point 9 is stored, subsequent rate calculations use the slope between points 2 and 3 as the baseline for subsequent storage decisions because point 2 was last point that was stored normally by storage.

The deadband override period can have any value and is not related to the the real-time window value.

Determining If the Real-Time Window Is Configured Appropriately for All Tags

To determine if the real-time window is configured correctly for a swinging door deadband, look at the number of data values that are forced to be stored while the system waits for the next valid data point to make the filtering calculation.

The SysRateDeadbandForcedValues system tag counts the number of "extra" points forced to be stored as a result of an insufficient real-time window for swinging door storage. Also, you can determine the number of points forced to be stored for an individual tag by executing a query that uses the full retrieval mode and specifies a quality detail of 2240, which indicates that these points were stored because of an insufficient real-time window.

If you find a large number of forced storage points, you can either reconfigure the tag to use delta storage or increase the real-time window.

Note The first two points received for a tag configured for swinging door storage are always stored.

Disk Requirements and Performance Considerations for a Swinging Door Deadband

One of the benefits of using the swinging door deadband is better data compression. However, because the storage system already provides a good compression ratio, the amount of disk space that is saved by applying this type of deadband for slow-changing tags (changing less than twice in a 15-minute interval) is negligible. For example, a tag that changes 12 times per hour will use 2K bytes of disk space in a 24-hour period. Even if only every fifth point is stored, the savings is only 1.5K bytes per day.

Also, use caution when setting the real-time window to accommodate a swinging door deadband.

- If your system has a large tag count or high data throughput, increasing the real-time window will increase the memory requirements for storage, because the storage system will have to process more data as real-time data, which is more resource-intensive than the storage of late or old data.
- If you increase the real-time window and you apply a swinging door deadband to a slow-changing tag, the amount of storage space required increases because the tag value is forced to be stored more often than if you used delta storage with no deadband.

Cyclic Storage

Cyclic storage is the storing of analog data based on a time interval. Cyclic storage writes a record to history at the specified interval, only if a data changes during the time interval. For example, you could store the value of an analog tag every five seconds.

The time interval for cyclic storage is called the storage rate. Each analog tag has its own storage rate. The storage rate you should select depends on the minimum timestamp resolution with which you want to be able to retrieve the data sometime in the future. Storing data using a very low time interval will result in a very high resolution of data being stored over a small time span. Storing data using a very high time interval, however, may result in significant data values being missed. An exception to this will be values received or generated due to a connect or disconnect event.

Available storage rates for analog tags are:

- 1, 2, 3, 5, 6, 10, 15, 30 seconds
- 1, 2, 3, 5, 6, 10, 15, 20, 30 minutes
- 1 hour

The timestamp for the first data value received during the cyclic time span will be used. For example, you might have specified a 10 second storage rate for a particular analog tag. The last data value received during the 10 second lapse will be stored, along with the corresponding timestamp.

The storage subsystem physically stores cyclically stored values in the same manner as it does value stored by delta. That is, it keeps track of repeated values in time in a logical manner and will "fill in" the missing values upon retrieval. For example:

1 1 1 2 2 2 3 3 4 4 4 5 5 5 ⟵ Data values from I/O Server.

1 1 2 3 4 4 5 ⟵ Cyclic storage method applied.

1 2 3 4 5 ⟵ Actual values stored on disk.

1 1 2 3 4 4 5 ⟵ Duplicate values "filled in" upon retrieval.

Data Conversions and Reserved Values for Storage

The Wonderware Historian can store values up to 32-bits. If the data source sends a 64-bit real number to Wonderware Historian, the number is converted to a 32-bit real number and, consequently, loses precision to six or seven decimal places. (The range depends on the number of binary digits, which does not exactly map to a fixed number of decimal places.) As a result, values that differ in the data source system beyond the seventh digit will not register as a change for delta storage and will not be re-stored. For example, a value of 1.0449991 followed by a 1.0450001 would both round to 1.045000, and the second point would not be stored.

Reserved values for storage are:

- -2,147,483,648, which represents an inserted NULL value for 32-bit tags.
- -32767, which represents an inserted NULL value for 16-bit tags.
- The last stored value for all types of tags.
- The last received value for analog tags stored cyclically.
- The last received value for tags that are configured for swinging door storage.
- Invalid values for all tag types except for variable length strings.

The maximum length of a string value that can be stored is 513 characters. Any characters over this limit are truncated. This limit does not apply to variable-length strings.

History Blocks

The Wonderware Historian historizes data in sets, or blocks, of files on disk. A history block is essentially a sub-folder of the main historian data storage folder, defined during installation or by subsequent dynamic configuration changes. A history block is self-contained, containing all the information necessary to retrieve data for the period represented by the history block. You specify the duration of history blocks. The default duration of a history block is one day, and the minimum allowed duration is one hour. The historian automatically creates a new history block at system startup, at scheduled block changeover times, at your request, or in response to certain dynamic configuration actions.

Note Configuration data and event history are not stored in the history blocks; they are stored in the Runtime database file.

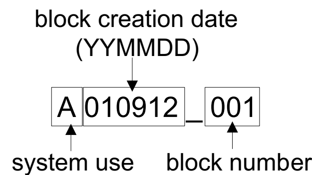
As data is acquired from the plant, the size of these history blocks grows on a continual basis, being limited only by the size of the hard disk on which the historian resides.

By storing plant data in the history blocks instead of in normal SQL Server tables, the historian can store data in a fraction of the space that would be required by a normal relational database. Compact storage formats reduce the storage space requirements than would be required in a standard relational database. Upon retreival, historical data is presented by the Wonderware Historian OLE DB provider as if it were stored in SQL Server tables.

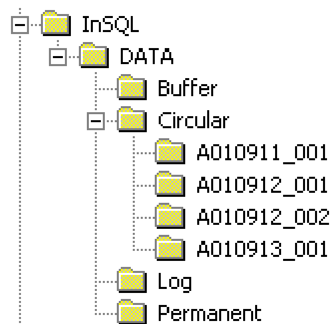
History Block Notation

Each history block is contained in a single sub-directory in the circular storage directory. The sub-directory name includes the date stamp of the Wonderware Historian computer at the time the block was created. Also, a numerical suffix is used to differentiate blocks of historical data created on the same day as a result of, for example, multiple restarts of the system, or where the block time interval configured is less than one day. The block duration can be configured by changing the *HoursPerBlock* system parameter. For more information on editing system parameters, see "Editing System Parameters" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide.*

In the following example, the "010912" portion of the directory name is the date that the history block was created. The "_001" portion is the numerical suffix that identifies this history block as the first block created that day.



By default, history blocks are 24 hours in size. In the following example, the system was restarted on September 12, 2001, resulting in two blocks existing for that date. The two blocks together span the 24 hours for that period.



History Block Creation

A single history block is created at system startup. After that, new history blocks are automatically created upon expiration of a designated time interval as stipulated in the database. The default value is 24 hours. You can change the time interval using the *HoursPerBlock* system parameter. For more information on editing system parameters, see "Editing System Parameters" in Chapter 9, "Viewing or Changing System-Wide Properties," in your *Wonderware Historian Administration Guide.*

You can manually start a new history block at any time by either:

- Executing a menu command from within the System Management Console. For more information, see "Starting a New History Block" in Chapter 5, "Managing Data Storage," in your *Wonderware Historian Administration Guide*.
- Using the **xp_NewHistoryBlock** extended stored procedure. For more information, see "xp_NewHistoryBlock" in Chapter 4, "Stored Procedures," in your *Wonderware Historian Database Reference*.

A new block is also created automatically if any of the storage data files becomes larger than 1.5 GB.

Note The system supports 999 history blocks per day. If this limit is reached, the system begins overwriting data in the first block for the day. A warning message will be issued in the Wonderware Historian Console when the limit is about to be reached. To reduce the number of history blocks created per day, increase your standard history block size (if it is less than 24 hours).

History Block Storage Locations

There are four types of storage locations for history blocks: circular, alternate, buffer, and permanent. The paths to the circular, buffer, and permanent storage locations are initially defined during installation. The alternate storage location can be defined later using the System Management Console.

Certain restrictions apply when specifying a path to the storage location. The circular storage location must be a local drive on the server machine, and the path must be specified using normal drive letter notation (for example, c:\Historian\Data\Circular). For a tier-1 historian, the alternate, buffer, and permanent storage locations can be anywhere on the network. For a tier-2 historian, the buffer and permanent storage locations can be anywhere on the network, but the alternate storage location must be on a local drive. The ArchestrA service user must have full access to network locations. The locations must be specified using UNC notation. Mapped drives are not supported.

When planning your storage strategy, be sure to allow enough disk space for storing your plant data for the required length of time.

Circular Storage Location

Circular storage is used for the main historical data storage. When the storage subsystem starts for the first time, the first history block is created, and data starts being written to that history block. The block of historical plant data is saved as a subdirectory in the circular storage directory.

The circular storage location consists of a single location, written to in a "circular buffer" fashion. When the free disk space on the disk containing the circular storage location drops below a minimum threshold or when the data is of a specified age, the oldest data is deleted out of this storage location and replaced with the new data to be stored. You can also limit the size of the circular storage location. When the contents of the circular storage location reach or exceed this limit, the oldest data will be deleted. "Automatic Deletion of History Blocks" on page 117.

Instead of data being deleted from the circular storage location, it can be moved into the alternate storage location, if this location is defined. As long as the free disk space for the circular storage location is below the threshold, the oldest data will be continuously migrated to the alternate storage location to make room for the new historical data.

It is the responsibility of the system administrator to monitor disk space and back up history blocks to storage media (such as DAT tape) on a periodic basis.

Alternate Storage Location

When the free disk space in the circular storage location goes below the defined threshold, the circular directory exceeds the specified maximum size, or the blocks reach a certain age, the storage subsystem will start moving the oldest history blocks to one or more alternate locations, if defined.

History blocks in the alternate storage area are managed in the same way as the blocks in the circular storage area. However, blocks will not be deleted based on age until the sum of the specified ages for both the circular and alternate storage has passed.

Note Only one alternate storage location is supported for this release.

Alternate storage locations are numbered. A block of data moves sequentially through the alternate locations until it is finally moved to the end of the last alternate location space, at which point the data is deleted from the system.

At a minimum, the alternate storage location must reside on a different logical drive than the circular storage location. A separate partition or physical drive would be better; a separate system is highly recommended. This storage location is optional.

Permanent Storage Locations

Permanent storage locations are used to store critical data (for example, reactor trips) that must not be overwritten. The storage subsystem will never attempt to delete data in this location. Data in a permanent storage location can be accessed and viewed along with the data stored in the circular storage location.

Use the **xp_DiskCopy** extended stored procedure to move history blocks to this storage location. For more information, see "xp_DiskCopy" in Chapter 4, "Stored Procedures," in your *Wonderware Historian Database Reference*.

Buffer Storage Locations

Buffer locations are used for temporary purposes, such as retrieval from a data archive. This storage location can reside on the same hard disk as the circular storage location or on a different disk. Data stored in the buffer storage location can be accessed and viewed along with the data stored in the circular storage location. Data is never deleted from this location by the storage subsystem.

Automatic Deletion of History Blocks

History blocks in the circular and alternate storage locations may be automatically deleted to make room for new history blocks. Whether or not the blocks are deleted is determined by the minimum threshold and the maximum size and/or age specified for the storage location.

The Configuration Service will check for available space in the circular and alternate locations if it detects any changes made by other subsystems or the user in controlled directories. If the realtime storage service is running, this check is performed every 20 seconds (which is the update time of the block.inf file).

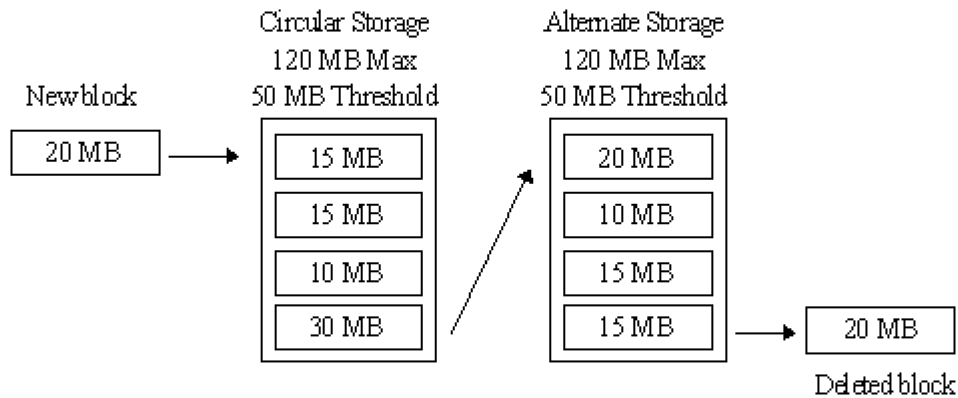
When the Configuration Service computes the sum of the sizes of all history blocks (including the current one) in the circular storage location and determines if there is enough space on the drive to hold all of the blocks.

If the space available on the storage location drive is below a certain threshold, the storage subsystem will delete enough of the oldest history blocks to bring the available disk drive space back to a positive value and then move the new history block in.

If an alternate storage location exists, the older block(s) will be moved there instead of being deleted. The alternate storage location functions exactly like the circular storage location. However, when the blocks exceed the set limits (minimum threshold, maximum size, or maximum age), the oldest blocks will be deleted from disk.

To avoid this loss of data, it is important that the system administrator regularly monitors the disk availability and periodically backs up old blocks to long term storage before they are deleted.

For example, a history block is stored in the circular storage location. The maximum size of the disk drive for circular storage is 120 MB. In addition to the circular storage location, an alternate location with a maximum disk drive size of 120 MB is defined. For both locations, the minimum threshold value is 50 MB. Essentially, this means that there is 70 MB of actual storage space.



Note The sizes in this example are purposely small; the disk drives for storage locations should be much larger.

You should typically set the minimum threshold to a value that is 1.5 times larger than the size of the biggest history block. This will provide the Configuration Service enough time to copy oldest history block from the circular location to the alternate, and then delete block from the circular location.

If you monitor the disk drive space available in the circular or alternate storage location over time, the value will fluctuate between the threshold value and the maximum size of the location, with sharp increases when blocks are moved out. While the system is moving a block(s) out, the space available will dip just below the threshold value before the increase.

If the maximum threshold is reached before the age of the block reaches the specified limit, the block is moved or deleted. A block will be moved or deleted within one history block duration of it reaching the age limit. If, for any reason, the system is unable to move a block that is past the age limit, the block will not be deleted until the size or space limit is reached.

About the Active Image

The active image is an allocation of memory in which copies of values of acquired data are temporarily held so as to service client requests while the actual data is being written to disk. The rate at which values are acquired into the active image depends on the rate of change of the incoming values for a particular tag. Values will collect in the active image until the default number of values to be held is reached. This value limit is initially set to 65 values (samples) per tag.

Note The rate at which values are acquired by the active image is NOT related to the value that is stored in the AcquisitionRate column of the Tag table.

When the sample limit for the active image is reached, the oldest tag values will start to be overwritten with new tag values. Overwriting the older tag values makes room for new tag values, with a default number of 65 values for each tag being held in the active image at any given time. Also, all of the values stored in the active image may not be stored to disk: it depends on the storage rate (from the StorageRate column in the Tag table). If the samples in the active image are acquired at a rate that is faster than the storage rate, tag values will be acquired into the active image at a higher resolution than they will be stored.

To prevent data values in the active image from being overwritten, the number of samples held would need to be increased to cover the time gap required for the storage subsystem to store the actual values to disk.

Automatic Resizing of the Active Image

The system continuously recalculates the optimum number of samples for each tag based on the data rates received. Beginning at one minute after startup and every five minutes thereafter, the system will perform a test to see if the amount of memory allocated for the active image needs to be increased. The system will calculate an average time span for a sample for a tag, based on the timestamp of the first and last samples in the active image and the number of samples. If necessary, the system will increase the number of samples. However, if the calculated time span is greater than 65 seconds, the system will not change the value from the default 65 values.

In short, if you receive more than 65 changes in a value within a 65-second interval, the system will increase the number of SamplesInActiveImage in the Tag table and increase the size in memory accordingly to accommodate the additional samples for the fast-changing tag values.

The new calculated sample number is the number of values required in the active image to hold data for 1 min (+15%), as calculated by the system. This value is updated only if the AIAutoResize system parameter is set to 1 and the number of required samples is greater than 65. This value is written to the SamplesInActiveImage column of the Tag table at system startup.

To turn off the automatic increasing of the active image, set the value of the *AIAutoResize* system parameter to 0. Also, you can change the rate at which the system recalculates the samples to be used for each tag in the active image using the *AIResizeSecInterval* system parameter. For more information, see "System Parameters" on page 33.

The number of samples in the active image never automatically decreases, but you could manually decrease it using the **ww_SetAISamples** stored procedure. However, the changes will not take effect until the next system startup; the running system will not automatically detect that the new number specified is lower than number of samples required to hold values in active image for one minute. For information, see Chapter 4, "Stored Procedures," in your *Wonderware Historian Database Reference*.

Resizing the active image is a very resource extensive operation. To prevent unnecessary "noise," after the active image adjusts the samples for a particular tag, it will change this number only if the increase in samples more than 10 percent of the previous number of samples.

You could use the active image to optimize your system, because the retrieval subsystem will first check the active image to see if the start time for all tags is in the active image. If so, only the active image is queried. If a tag does not exist in the active image at the start time, data is retrieved in parallel from both the active image and disk and then merged together. If duplicate values are encountered during the merge, the value from disk will be used.

For example, if you need to create hourly reports for a week of production based on the "SysTimeHour" tag, then you might want to increase the number of samples in the active image for that tag to 144 (24 * 7 = 144). This way, the system will never go to disk to retrieve data for "SysTimeHour," and both the overall time for retrieving data and the CPU load will decrease.

Important Although you can manually set the active image samples for a variable length string to a value other than 0, NULL, or 65, this is not recommended. The performance impact will be extremely high, because each sample for a variable length string is 1038 bytes allocated in memory.

How the Active Image Storage Option Affects Data Retrieval

You can configure whether to see the data in the active image as it comes into the system or based on the storage algorithm. You can specify this when you configure the storage options for the tag. For more information, see Chapter 2, "Configuring Tags," in your *Wonderware Historian Administration Guide*.

If you configure the active image to hold all received values, you may see a discrepancy between tag values for the same time period, depending on when you ran the query. This is true for tags that are stored cyclically or by exception (delta) with value or time deadbands.

For example, if you run a query for data between 20011206 1:00:00:000 and 20011206 1:02:00:000, and data during this time period is being held in the active image, data will be returned at the resolution it was acquired.

However, if you run the same query later, and the values are now stored in the history blocks on disk, you may not see such a high resolution of data. Some values may be discarded if the storage rate is slower than the acquisition rate. It may appear that you have "lost" some of the values when, in fact, they were never configured to be stored in the first place.

Dynamic Configuration Effects on Storage

During dynamic configuration, the Configuration Service will determine if the changes require the creation of a new history block.

The storage of existing tags will not be interrupted during dynamic reconfiguration; however, storage for new or affected tags may not begin until five to ten minutes after the reconfiguration was committed, unless you have already allocated memory for the new tags. For more information, see "Pre-allocating Memory for Future Tags" in Chapter 2, "Configuring Tags," in your *Wonderware Historian Administration Guide*.

If the storage subsystem cannot complete the reconfiguration for some reason, a critical or fatal error is generated and written to the error log. An error is also generated if the reconfiguration process takes so long that the storage buffer overflows and data is lost.

Memory Management for Data Storage

By default, the Wonderware Historian loads all tag information for the history blocks into memory so that it can more efficiently service requests for data. This tag information includes tag properties and indexing information and allows for quick navigation through the files containing real-time data. The process that manages the tag information in memory is the Wonderware Historian Indexing Service (aahIndexSvc.exe).

For large systems, it is possible that loading the tag information from all of the history blocks will require more memory than the 2 GB limit that is imposed by the Windows operating system for a single process. The actual limit may be even be less than 2 GB, if the amount of installed RAM is insufficient.

The total amount of tag information for the history blocks depends not only on the total number of tags, but also on the number of tag versions, which are created during modifications to old data. Therefore, it is recommended that you monitor the memory consumption for all systems, large and small, if you are regularly performing data inserts, updates, or CSV file imports.

To avoid excessive memory consumption by the Wonderware Historian Indexing Service, tune and monitor the service for your system using the following system parameters and system tags.

- *HistoryCacheSize* and *HistoryDaysAlwaysCached* system parameters

  You can limit the maximum amount of memory the Indexing Service can use for tag information by adjusting the value of the *HistoryCacheSize* system parameter. When this parameter is set 0 (default), the Indexing Service selects a default cache value automatically by taking into account the amount of installed physical memory and the maximum available address space for the process. In some rare cases when some specific performance tuning is needed, you may want to set the HistoryCacheSize parameter manually. In this case, the Indexing Service uses the specified value, but still may automatically change the effective HistoryCacheSize if the specified value is too low or two high.

  Regardless of whether the effective HistoryCacheSize was selected automatically (default) or specified by you, the Indexing Service manages the cache using a "least-recently used" algorithm. In this algorithm, when there is a request to access a history block that is not currently cached, the Indexing Service unloads the tag information from the least-recently used history block and then loads the tag information from the requested block.

  All of these operations are performed automatically in the background, but you may notice a slowness data retrieval if the data is retrieved from a block that is not currently loaded into memory. Keep in mind that the smaller the amount of memory that you allocate for the cache, the potentially longer it may take to service data requests.

  To guarantee the maximum retrieval performance for the newest history blocks (for example, if you a running a trend application for the last week), you can "lock" a certain number of the most recent history blocks in the cache. To do this, set the number of days to be locked in the cache by changing the *HistoryDaysAlwaysCached* system parameter.

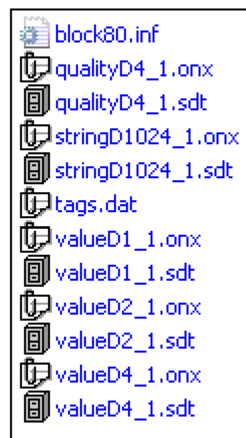
- SysHistoryCacheFaults and SysHistoryCacheUsed system tags

  To determine if you need to clamp the memory used by the Indexing Service, use the Windows Task Manager application or the Performance console to see how much memory is used by the aahIndexSvc.exe process. Also, you can monitor the SysHistoryCacheFaults and SysHistoryCacheUsed system tags. A high number of cache faults may be indicating that the cache size is insufficient. The SysHistoryCacheUsed system tag shows the number of bytes currently used for keeping the tag information. This tag may be helpful to see how much memory is consumed by the tag information, even if the memory management is not enabled.

At any time, you can observe the current status of the history blocks in the Wonderware Historian Management Console. When the tag information from a history block is not loaded into memory, the history block icon is dimmed. You can manually refresh the console window to see changes in the status for the history blocks.

About Snapshot Files

Each history block consists of a set of "snapshot" files (.sdt) in which the data values are actually stored.



Note Replication will add some additional files in this folder.

The snapshot file notation is as follows:

<*type*>D*X_x*

where,

<*type*> = "value" for discrete and analogs or "string" for strings.

X = The byte size of the value. For strings of fixed length that are 128 bytes or less, this value will be the actual length in bytes. For strings of fixed length that are more than 128 bytes, this value will be 1024.

_x = The number for the data "stream." For a snapshot file storing original real-time data, the value will be 1. To handle other data, such as for inserts of old data and updates, the storage subsystem may create additional snapshot files to hold the values, instead of inserting them into the initial snapshot file. By using multiple snapshot files, the storage subsystem can store the different parallel data streams simultaneously and more efficiently.

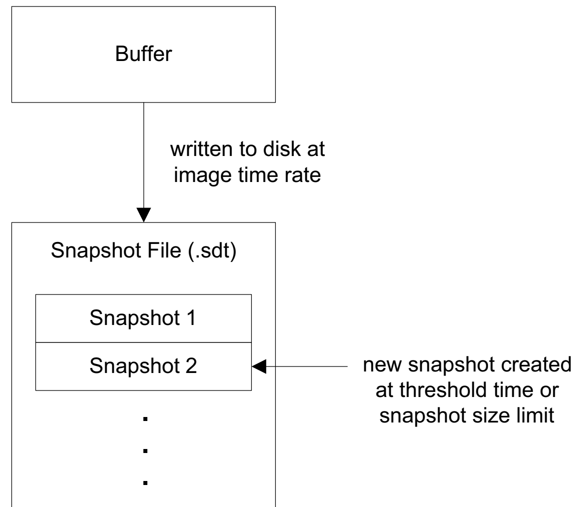
A snapshot file exists for each byte size of data values:

Snapshot	Description	Value type stored
-1	Used to store values other than 1-bit, 16-bit, or 32-bit values.	Fixed-length strings, variable-length strings
0	Used to store quality detail.	Quality detail
1	Used to store 1-bit values.	Discretes
2	Used to store 16-bit values.	Analogs, integers
4	Used to store 32-bit values.	Reals (floats), longs, and integers
8 (Future)	Used to store 64-bit values.	Integers

How Snapshot Files are Updated

Before tag values are stored to disk in snapshot files, they collect in memory in dynamically allocated buffers. (These buffers are different than the active image.) Values in the buffers are stored to snapshot files according to the image time rate, which is every 30 seconds by default.

Within each snapshot file, there are sets of value snapshots. A snapshot contains a copy of all of the stored tag values at a certain point in time.



When the system is first started, the first snapshot is generated in the .sdt file. This snapshot contains the initial values for the stored tags. Every 30 seconds, new values for each tag are retrieved from the snapshot buffer and are added to the current snapshot.

When the size of the current snapshot reaches a particular limit (by default, 2 MB) or when an hour has elapsed, a second snapshot is generated. When the size limit for the second snapshot is reached or an hour has passed, a third snapshot is created, and so on.

Every 30 seconds, the system checks how many values are collected for each tag. If the number exceeds 15000, a new snapshot is automatically generated.

Chapter 6

Data Retrieval Subsystem

The Wonderware Historian data retrieval subsystem 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 standard 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 retrieval subsystem includes:

- An implementation of a SQL Server data provider, which determines whether the requested data is saved in normal SQL Server tables or in history blocks.
- A retrieval service, which is responsible for extracting the requested data from the history blocks and presenting to the Wonderware Historian OLE DB provider as "virtual" history tables.
- A set of SQL Server extensions, which 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 Chapter 5, "Data Storage Subsystem."

Data Retrieval Components

The following table describes the components of the data retrieval subsystem.

Component	Description
Runtime database	SQL Server database in which configuration and event data are stored.
History Blocks	Files in which plant history data is stored. In the context of Microsoft SQL Server, history blocks are considered a non-local data source.
Retrieval Service (aahRetSvc.exe)	Process that retrieves data from the history blocks and presents it as data sets. This process runs as a Windows service.
Manual Data Acquisition Service (MDAS)	Component that allows data retrieval, data insertions, and configuration functions, such as tag creation.
Wonderware Historian OLE DB Provider	A SQL Server software component used to query data in history blocks. The Wonderware Historian OLE DB provider can expose history data to client applications as if it were formatted as normal SQL Server tables.
Wonderware Historian Time Domain Extensions	Special Transact-SQL syntax extensions that allow for increased retrieval functionality for history data.
Wonderware Historian I/O Server (aahIOSvrSvc.exe)	Internal process that allows clients to access current tag values from the active image using the SuiteLink or DDE protocols.
Query Application(s)	Either a command-line application or point-and-click query tool that can connect to the Microsoft SQL Server and that uses Transact-SQL statements to retrieve data.

For a complete diagram of the Wonderware Historian architecture, see "Wonderware Historian Subsystems" on page 19.

Data Retrieval Features

Some of the main features of the data retrieval subsystem are:

- All tag types can be included in the same query when retrieving from the History table. Any combination of tags can be submitted in a single query.
- Both fixed length and variable length strings are supported.
- All internal time computation and manipulation is done using the Win32 FILETIME type. The resolution of FILETIME is 100 nano-seconds. The resolution exposed in queries depends on the version of SQL Server used.
- All times are handled internally as absolute time (UTC). Conversions to and from local time are handled going in and out of retrieval so the external interface is local time.
- Non-real-time data is supported (for example, store-and-forward data or data imported from a comma-separated values (CSV) file.
- Retrieval of different versions is supported.

History Blocks: A SQL Server Remote Data Source

Remote data sources 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." In the case of the Wonderware Historian, a remote data source is the set of history block files. All tag data is stored in history blocks. For more information on history blocks, see "History Blocks" on page 113.

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 Service

The retrieval service (aahRetSvc.exe) retrieves history data from both the active image and history blocks on disk. The retrieval service:

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

Linear scaling for analog tags is performed within retrieval using the following formula.

V_out = (V_in - MinRaw)*(MaxEU - MinEU)/(MaxRaw - MinRaw) + MinEU

where:

V_out = scaled output value

V_in = stored raw value

MinRaw = minimum raw value for the tag

MaxRaw = maximum raw value for the tag

MinEU = minimum engineering unit value

MaxEU = maximum engineering unit value

About the Wonderware Historian OLE DB Provider

Object Linking and Embedding for Databases (OLE DB) 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.

The benefit of using OLE DB is that it 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, such as from a SQL Server database, an Oracle database, and a Microsoft Access database. A query that accesses data from multiple, dissimilar data sources such as these is called a "heterogeneous query," with "heterogeneous" meaning "dissimilar." A heterogeneous query can also be called a "distributed query," because the data can be distributed across various data sources.

SQL Server uses OLE DB to process heterogeneous queries and makes linking data between the data sources much 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 Wonderware 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 Wonderware Historian OLE DB provider has a rich set of query capabilities.

The name of the Wonderware Historian OLE DB provider is "INSQL." The Wonderware Historian OLE DB provider is installed during Wonderware Historian installation and then associated, or linked, with the Microsoft SQL Server. For information on the syntax for linking the Wonderware Historian OLE DB provider, see "Linking the Wonderware Historian OLE DB Provider to the Microsoft SQL Server" on page 146.

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

When you execute a query and specify the Wonderware Historian OLE DB provider in the syntax, the Microsoft SQL Server parser will pass the appropriate parts of the data request to the Wonderware Historian OLE DB provider. The Wonderware 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 Wonderware 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 Wonderware Historian OLE DB provider. The Wonderware Historian OLE DB provider is SQL-92 compliant.

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

Extension (Remote) Tables for History Data

Some of the history tables are rowset representations that provide a means for handling acquired plant data. These tables are not part of normal SQL Server functionality. A normal SQL Server table stores data directly in the database's data device file (.mdf). An extension table, however, presents data as if it were a real table, but it does not physically exist in the data device. An extension table is a logical table that is populated from other types of data files; thus, the data is stored "remotely" from SQL Server. In the case of the Wonderware Historian, the data files are the history blocks generated by the storage system.

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 Wonderware Historian OLE DB provider in the syntax of the query.

The extension tables are:

- AnalogSummaryHistory (INSQL.Runtime.dbo.AnalogSummaryHistory)
- History (INSQL.Runtime.dbo.History)
- HistoryBlock (INSQL.Runtime.dbo.HistoryBlock)
- Live (INSQL.Runtime.dbo.Live)
- StateSummaryHistory (INSQL.Runtime.dbo.StateSummaryHistory)
- StateWideHistory (INSQL.Runtime.dbo.StateWideHistory)

The AnalogHistory, DiscreteHistory, StringHistory, AnalogLive, DiscreteLive, StringLive, AnalogWideHistory, DiscreteWideHistory, StringWideHistory, and v_SummaryData tables are provided for backward compatibility. For more information, see Chapter 6, "Backward Compatibility Entities," in your *Wonderware Historian Database Reference.*

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

For more information on the history extension tables, see "History Tables" in Chapter 1, "Table Categories," in your *Wonderware Historian Database Reference.*

Query Syntax for the Wonderware Historian OLE DB Provider

The most common Wonderware 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 Wonderware Historian OLE DB provider to retrieve history data:

- Using the Four-Part Naming Convention
- Using a Wonderware Historian OLE DB Provider View
- Using the OPENQUERY Function
- Using the OPENROWSET Function

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 Wonderware Historian OLE DB provider is known to the Microsoft SQL Server. In order for a query to be passed on to the Wonderware 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 Wonderware 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 Wonderware 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 Wonderware 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 Wonderware 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 Wonderware 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 Wonderware 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 a Wonderware 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 Wonderware Historian OLE DB provider. Consequently, only the syntax that the Wonderware 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 "Using Variables with the Wide Table" on page 312.

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"
       ')
```

Syntax Options Supported

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

Wonderware Historian OLE DB Provider Unsupported Syntax and Limitations

The Wonderware 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 Wonderware 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 Chapter 1, "Table Categories," in your *Wonderware 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" on page 281.

OR Clause Limitations

You cannot use the OR clause to specify more than one condition for a time domain extension. For more information, see "Wonderware Historian Time Domain Extensions" on page 147.

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 work-around 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 Wonderware 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 "Using an INNER REMOTE JOIN" on page 281.

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 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 vValue column in an OPENQUERY statement. If you are using OPENQUERY on the History table, you must filter on the vValue column outside of the query.

In the following example, the value of the vValue 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 optomizer 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 intead 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.

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

Because the Wonderware 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 Wonderware 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 Wonderware Historian installation. If, for some reason, you need to re-link the Wonderware 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 Wonderware Historian OLE DB provider as the linked server. Use this name to specify the Wonderware 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 Wonderware 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).

Wonderware 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 Wonderware Historian. The Wonderware 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
- wwResolution
- wwRetrievalMode
- wwTimeDeadband
- wwValueDeadband
- wwEdgeDetection
- wwTimeZone
- wwVersion
- wwInterpolationType
- wwTimeStampRule
- wwQualityRule
- wwValueSelector
- wwStateCalc
- wwFilter

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 "Wonderware Historian OLE DB Provider Unsupported Syntax and Limitations" on page 137.

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.

Wonderware Historian I/O Server

The Wonderware Historian I/O Server (aahIOSvrSvc.exe) is the interface for clients to access data from the active image of a Wonderware Historian by the SuiteLink protocol. The Wonderware Historian I/O Server can update items with current values for given topics, providing "real-time" I/O Server functionality. Tag values in the Wonderware Historian I/O Server are acquired from the active image. For more information on the active image, see "About the Active Image" on page 119.

Note NetDDE is not supported.

The Wonderware Historian I/O Server is pre-configured with a single topic, **Tagname**. The Wonderware 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 Wonderware Historian I/O Server, a "hot" link is established between the client and the Wonderware Historian I/O Server. For more information on I/O Server addressing conventions, see "I/O Server Addressing" on page 74.

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

By default, the Wonderware 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 Wonderware Historian I/O Server from within the System Management Console. For more information on the System Management Console, see Chapter 1, "Getting Started with Administrative Tools," in your *Wonderware Historian Administration Guide*.

The Wonderware Historian I/O Server is a read-only server; clients cannot update data in the active image.

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

Chapter 7

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 a Wonderware 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.

A Wonderware Historian with a version earlier than 9.0 supports two retrieval modes:

- Cyclic Retrieval
- Delta Retrieval

A Wonderware Historian with a version of 9.0 or higher supports various additional modes:

- Full Retrieval
- Interpolated Retrieval
- “Best Fit” Retrieval
- Average Retrieval
- Minimum Retrieval
- Maximum Retrieval
- Integral Retrieval

- Slope Retrieval
- Counter Retrieval
- ValueState Retrieval

A Wonderware Historian with a version of 10.0 or higher supports the following additional mode:

- RoundTrip 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 Wonderware 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 Wonderware 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.

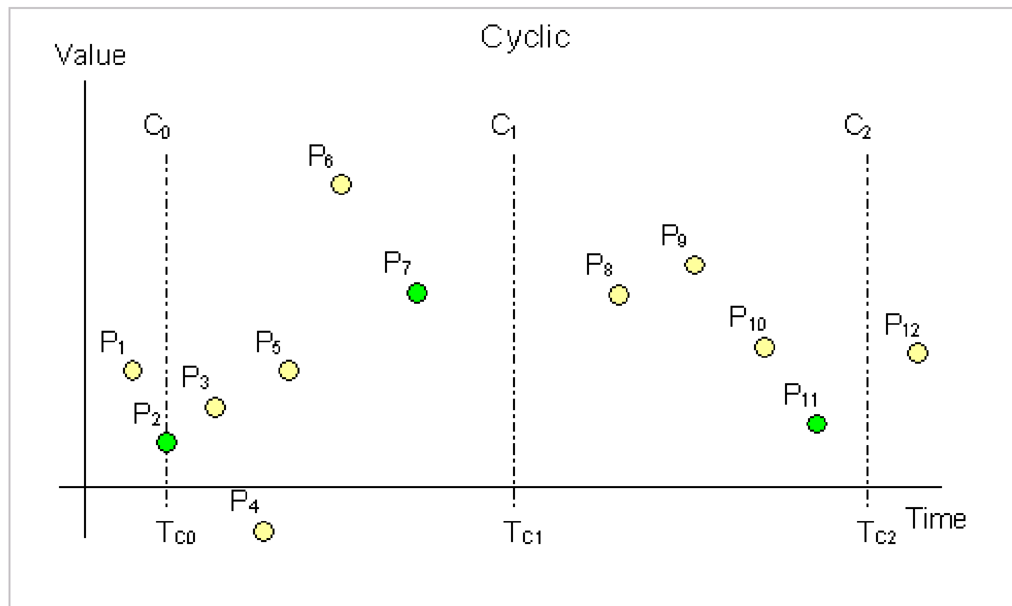
If no data value is actually stored at a cycle boundary, the last value before the boundary is returned.

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 on page 171.

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 219.
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222.
- History Version (wwVersion) on page 235.
- Timestamp Rule (wwTimestampRule) on page 240 (only on Wonderware Historian 9.0 and above).

Cyclic Retrieval - Query Examples

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

```
wwRetrievalMode = 'Cyclic'
```

Query 1

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 161.

Cyclic Retrieval - Handling NULL Values

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

Delta Retrieval

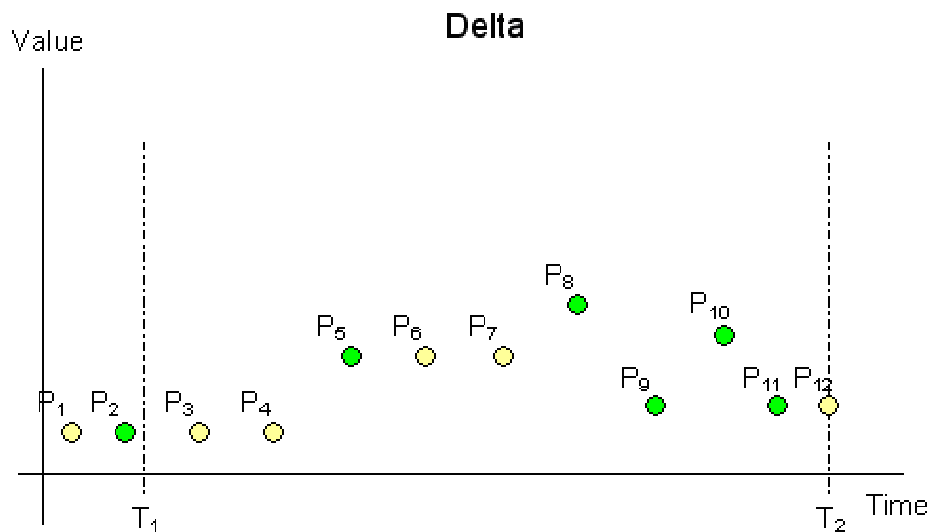
Delta retrieval, or retrieval based on exception, is the retrieval of only the changed values for a tag(s) for the given time interval. That is, duplicate values are not returned. It works with all types of tags.

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 227
- Value Deadband (wwValueDeadband) on page 231
- History Version (wwVersion) on page 235

Delta Retrieval - Query Examples

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

```
wwRetrievalMode = 'Delta'
```

Query 1

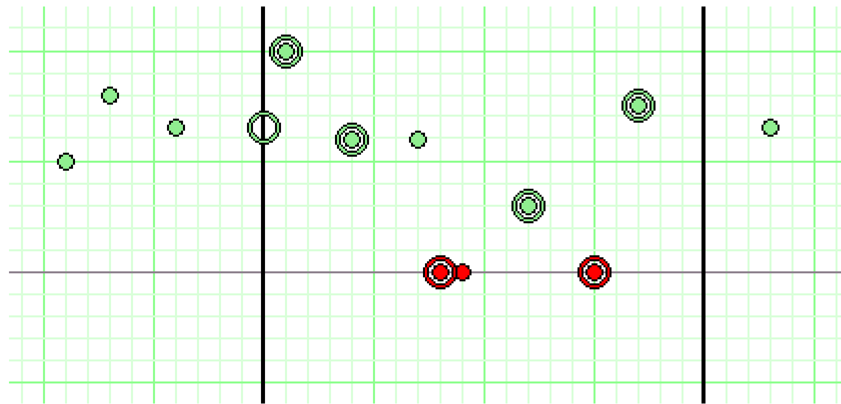
As an example of how delta mode works, consider the following query:

```
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 Using wwResolution, wwCycleCount, and wwRetrievalMode in the Same Query on page 285.

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 227 and Value Deadband (wwValueDeadband) on page 231.

Query 1

```
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
```

Query 2

```
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

Query 3

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
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

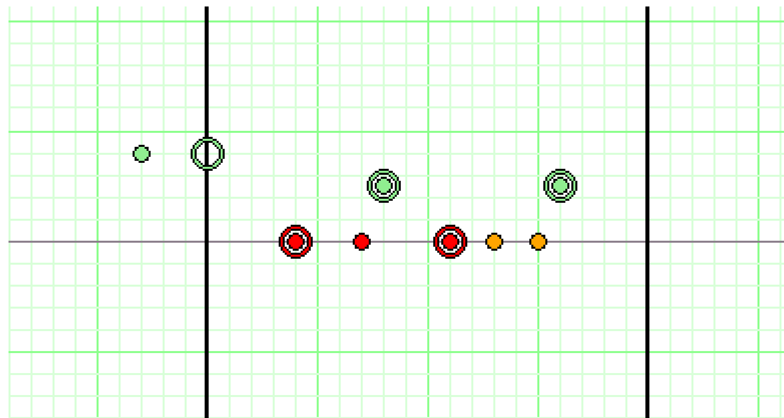
The initial NULL value after a non-NULL is always returned. Multiple NULL values are suppressed. The first non-NULL after a NULL is always returned even if it is the same as the previous non-NULL value.

```
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
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.

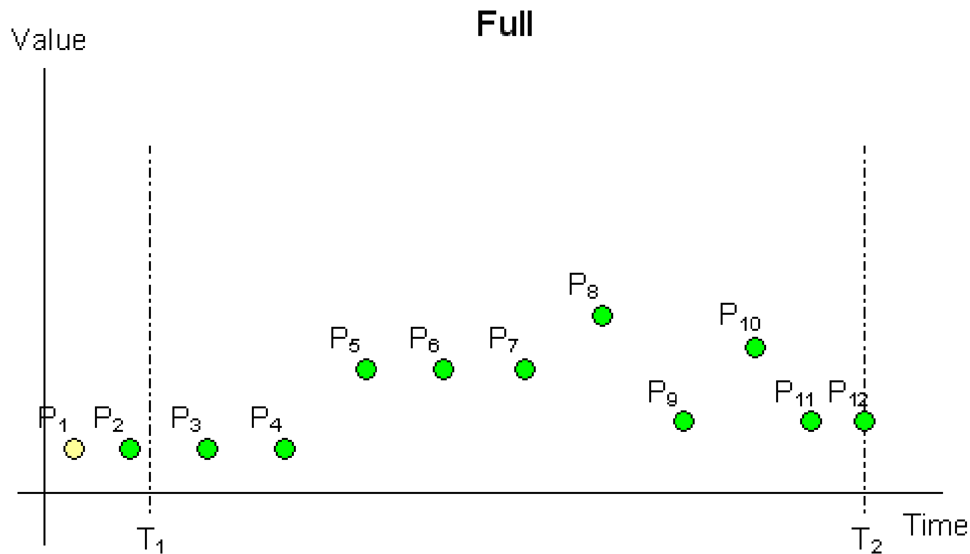
By using full retrieval in conjunction with storage without filtering (that is, no delta or cyclic storage mode is applied at the historian), you can retrieve all values that originated from the plant floor data source or from another application.

Full retrieval best represents the process measurements recorded by the Wonderware 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 235

Full Retrieval - Query Examples

Query 1

```
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 161.

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.

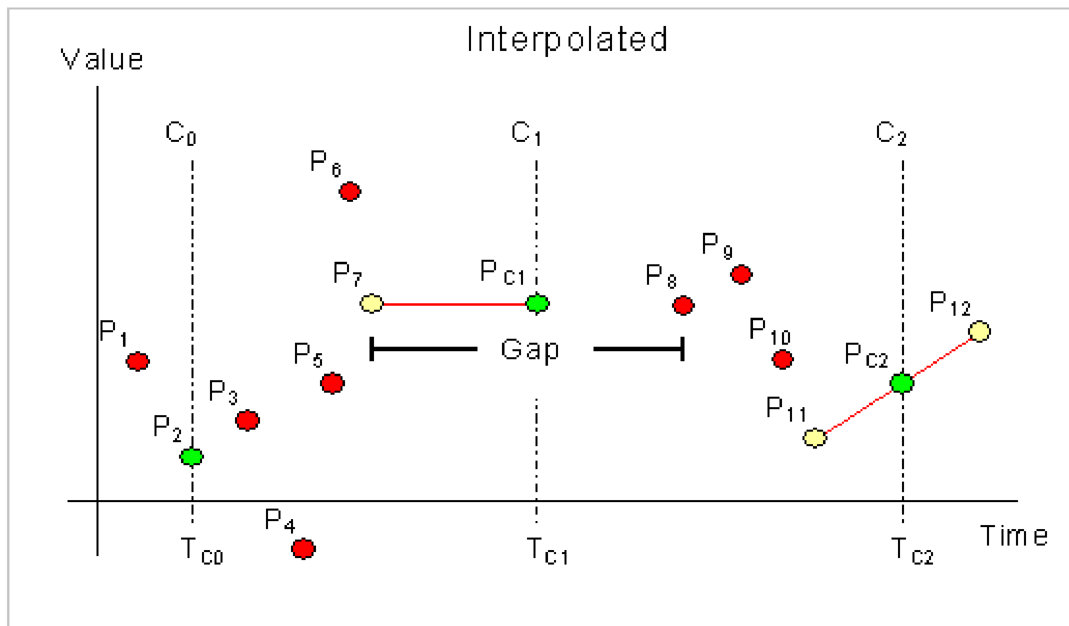
By default, interpolated retrieval uses the interpolation setting specified for the tag in the Wonderware Historian. This means that if a tag is set to use stair-step interpolation, interpolated retrieval gives the same results as cyclic retrieval.

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 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}. 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, P_{C1}, P_{C2}) 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 P_{C1} is returned, which is the NULL value represented by P_7 shifted forward to time T_{C1}. At the last cycle boundary, point P_{C2} is returned, which has been interpolated using points P_{11} and P_{12}.

Interpolated Retrieval - Supported Value Parameters

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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Interpolation Type (wwInterpolationType) on page 237
- Timestamp Rule (wwTimestampRule) on page 240
- Quality Rule (wwQualityRule) on page 244

Interpolated Retrieval - Query Examples

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

```
wwRetrievalMode = 'Interpolated'
```

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
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

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
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

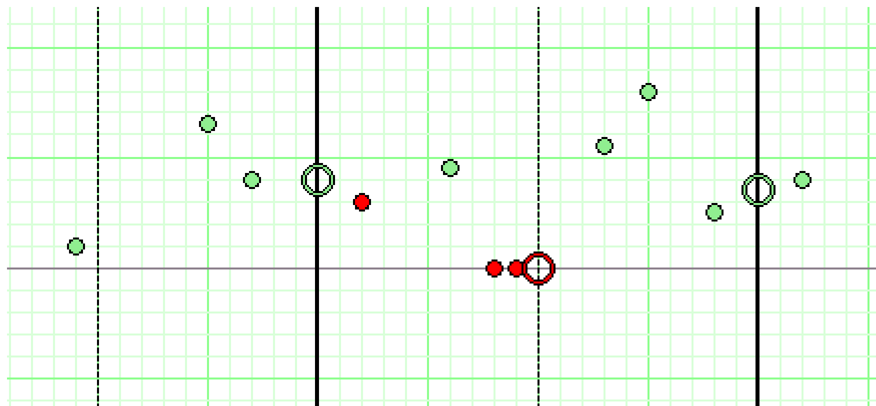
Query 3

```
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.

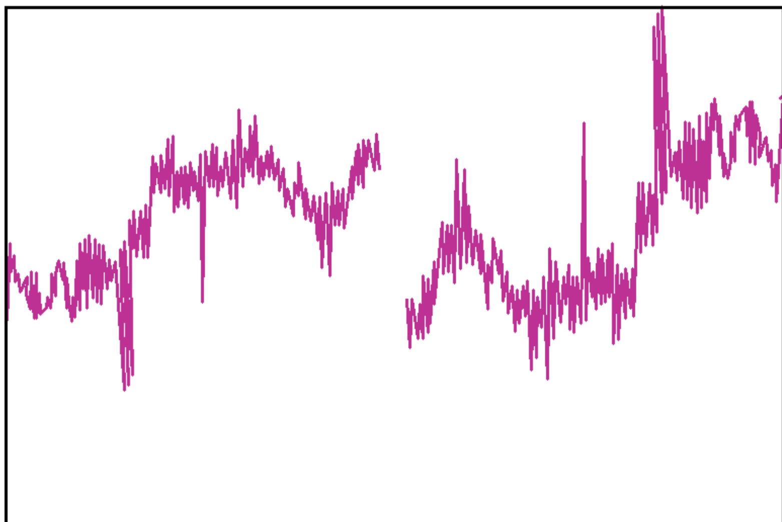
If a valid value precedes a cycle boundary, but is followed by a NULL value after the cycle boundary, no interpolation will be used and wwInterpolationType will be set to STAIRSTEP for that value.

"Best Fit" Retrieval

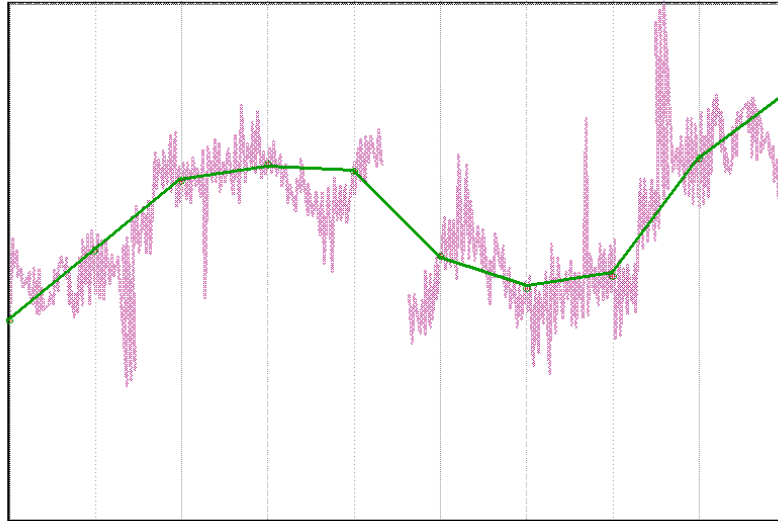
For the "best fit" retrieval mode, the total time for the query is divided into even sub-periods, and then up to five values are returned for each sub-period:

- 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)

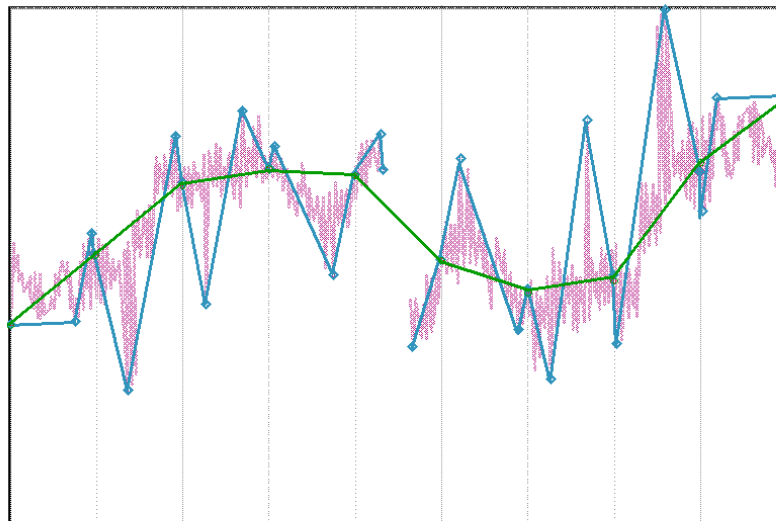
"Best fit" retrieval allows for a compromise between delta retrieval and cyclic retrieval. For example, delta retrieval can accurately represent a process over a long period of time, as shown in the following trend. However, to achieve this representation, a large number of data values must be returned.



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.

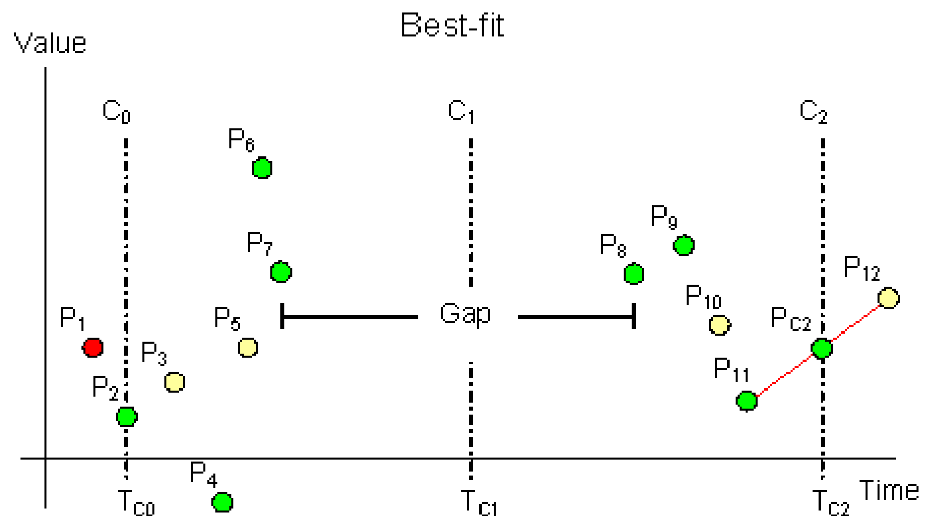
As in cyclic retrieval, the number of cycles is based on the specified resolution or cycle count. However, the number of values returned is likely to be more than one per cycle.

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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Interpolation Type (wwInterpolationType) on page 237
- Quality Rule (wwQualityRule) on page 244

Best Fit Retrieval - Query Examples

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

```
wwRetrievalMode = 'BestFit'
```

Query 1

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. The values of the initial and final points will be determined by interpolating the points preceeding 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 165.

Best Fit Retrieval - Handling NULL Values

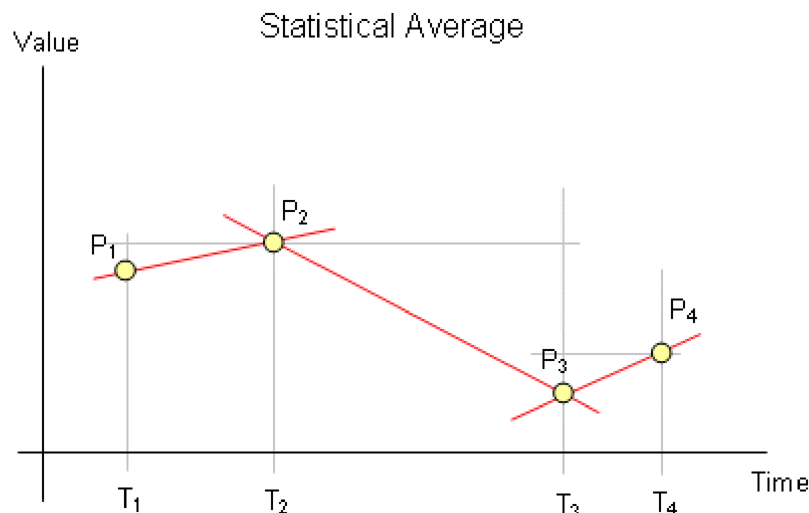
When any of the four good points are returned from a cycle that contains gaps or from an incomplete cycle with the query end time located inside of the calculation cycle the quality detail of each of the non-null points returned is modified to alert the user to this fact. This is done by performing a 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:

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

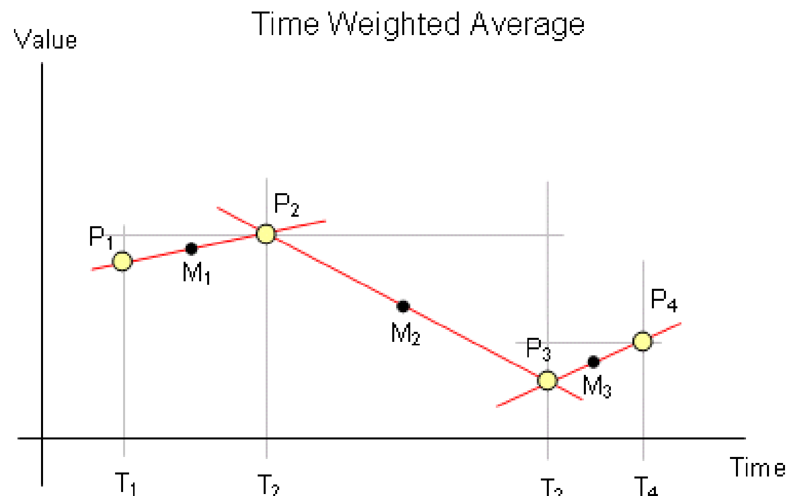


For a time-weighted average, values are multiplied by the time difference between the points to determine the time-weighted value. Therefore, the longer a tag had a particular value, the more weight that value holds in the overall average. The overall average is determined by adding all of the time-weighted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

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

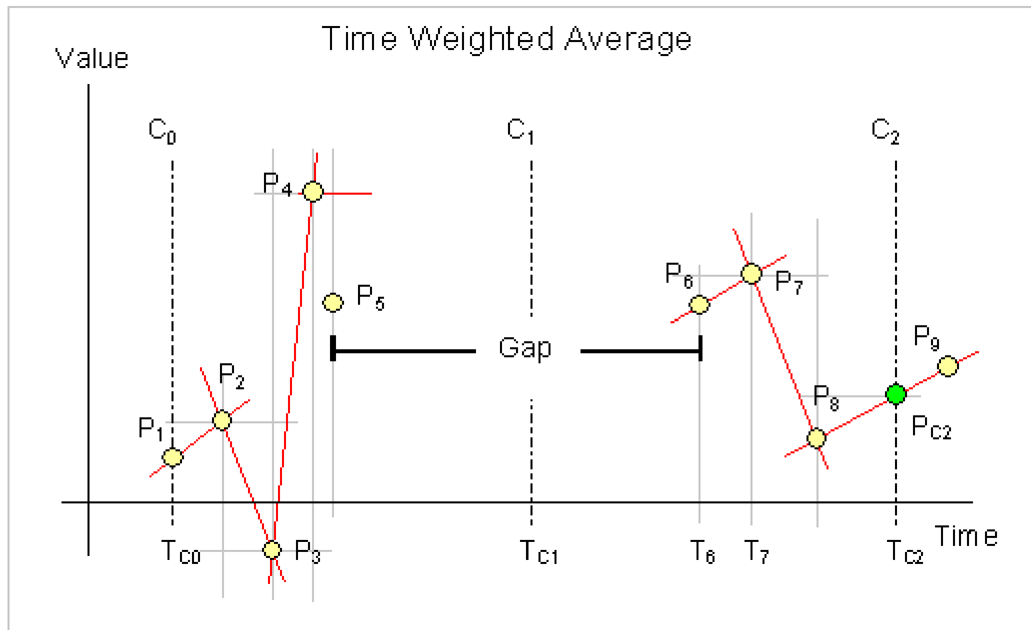
The SQL Server AVG aggregate is a simple statistical average. Using the average retrieval mode with a cycle count of 1 returns data much faster than the AVG aggregate, and usually more accurately due to the time weighting. The event subsystem also returns a simple statistical average.

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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Interpolation Type (wwInterpolationType) on page 237
- Timestamp Rule (wwTimestampRule) on page 240
- Quality Rule (wwQualityRule) on page 244

Average Retrieval - Query Examples

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

```
wwRetrievalMode = 'Average'
```

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

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

For an additional example, see Querying Aggregate Data in Different Ways on page 316.

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.

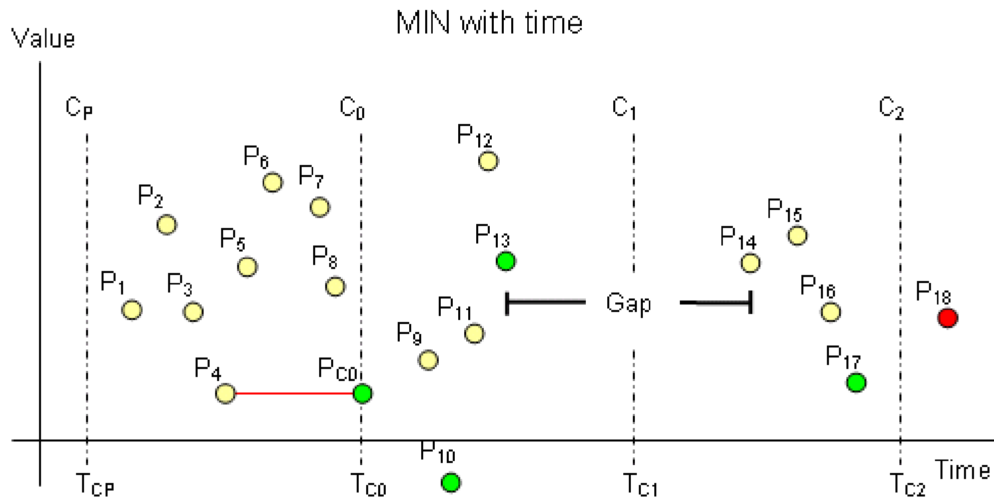
Minimum retrieval only works with analog tags. For all other tags, normal delta results are returned.

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.

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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Quality Rule (wwQualityRule) on page 244

Minimum Retrieval - Query Examples

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

```
wwRetrievalMode = 'Min'
```

or

```
wwRetrievalMode = 'Minimum'
```

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

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

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 161.)

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.

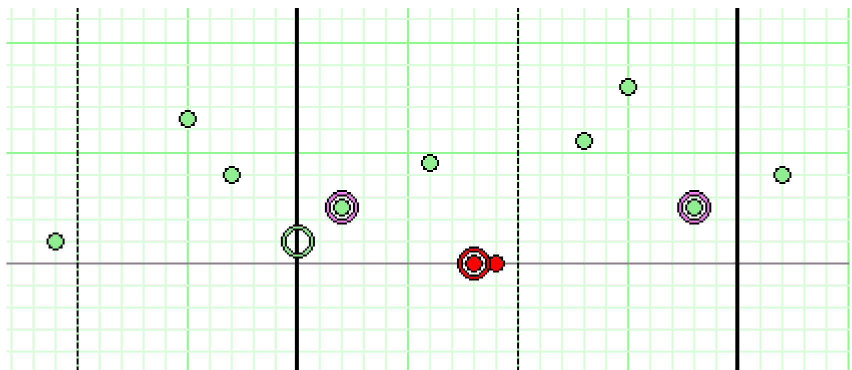
As an example of how minimum retrieval mode handles NULLs, consider the following query:

```
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.

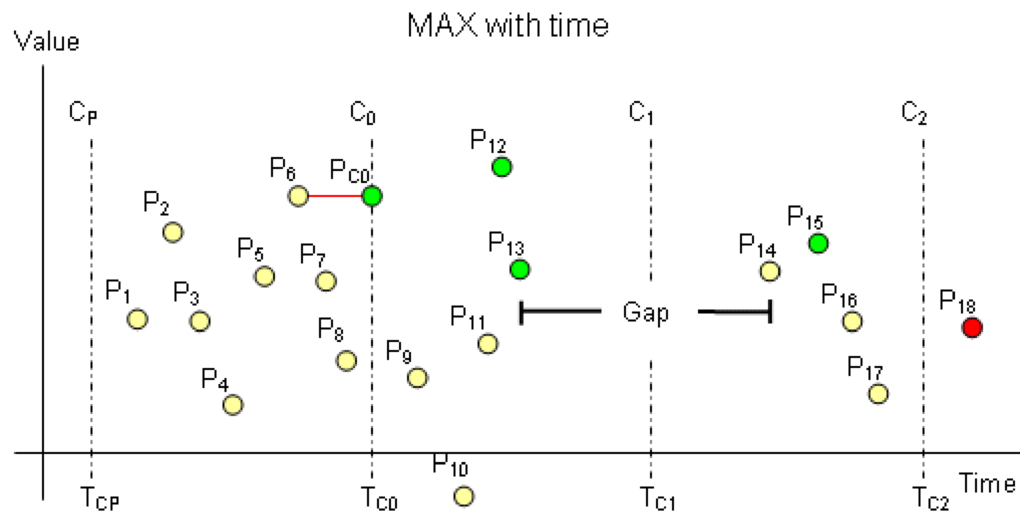
Maximum retrieval only works with analog tags. For all other tags, normal delta results are returned.

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.

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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Quality Rule (wwQualityRule) on page 244

Maximum Retrieval - Query Examples

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

```
wwRetrievalMode = 'Max'
```

or

```
wwRetrievalMode = 'Maximum'
```

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

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

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
(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 286.)

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:

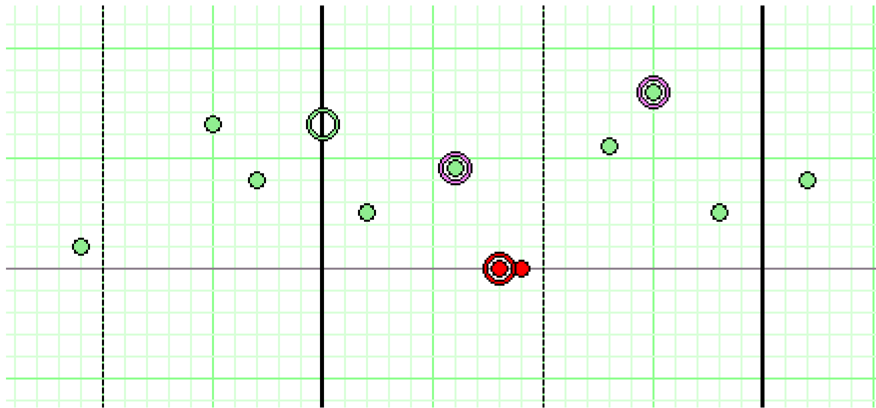
Tagname	DateTime	Value	QualityDetail
A001	2009-09-12 00:09	0.2	192
A001	2009-09-12 00:15	1.3	192
A001	2009-09-12 00:17	0.8	192
A001	2009-09-12 00:22	0.5	192
A001	2009-09-12 00:26	0.9	192
A001	2009-09-12 00:28	0.0	249
A001	2009-09-12 00:29	0.0	249
A001	2009-09-12 00:33	1.1	192
A001	2009-09-12 00:35	1.6	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 176.
- 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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Interpolation Type (wwInterpolationType) on page 237
- Timestamp Rule (wwTimestampRule) on page 240
- Quality Rule (wwQualityRule) on page 244

Integral Retrieval - Query Examples

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

```
wwRetrievalMode = 'Integral'
```

Query 1

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
(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 316.

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.

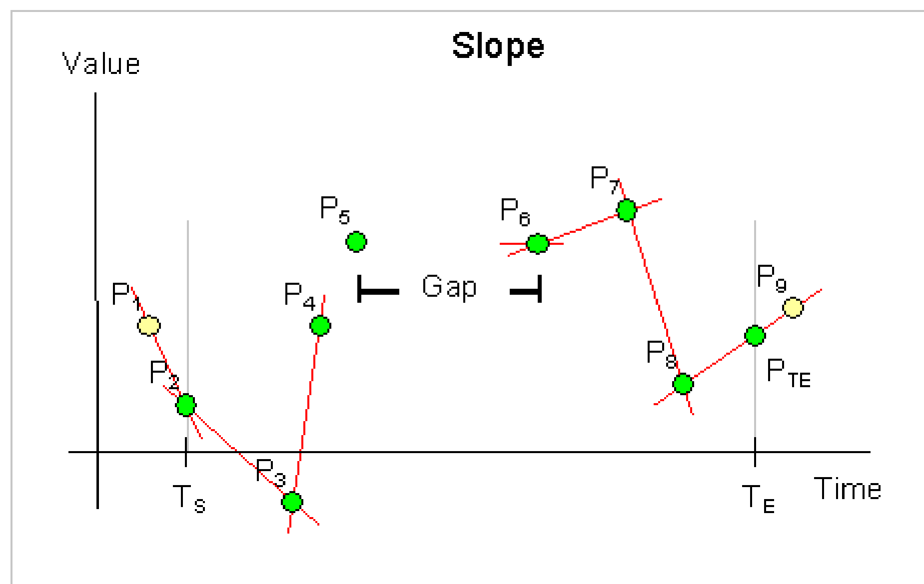
The slope retrieval mode can be considered a delta mode. Apart from the initial value and a value at the query end time, all returned points are calculated delta points returned with the timestamp of an actual delta point.

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 235
- Quality Rule (wwQualityRule) on page 244

Slope Retrieval - Query Example

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

```
wwRetrievalMode = 'Slope'
```

Query 1

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 286.)

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 - 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.

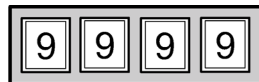
Slope Retrieval - Handling NULL Values

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 Wonderware 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.

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.

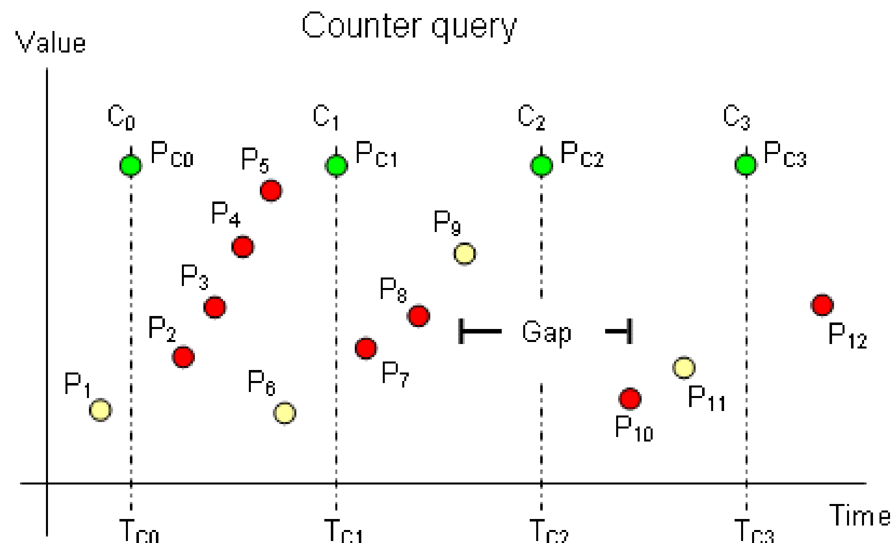
Counter retrieval is a true cyclic mode. It returns one row for each tag in the query for each cycle. The number of cycles is based on the specified resolution or cycle count.

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 244. When a rollover has occurred in the calculation cycle, a special quality detail of 212 is returned in all non-NULL cases.

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.

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 - 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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Timestamp Rule (wwTimestampRule) on page 240
- Quality Rule (wwQualityRule) on page 244

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 preceeding cycle. A NULL point with cycle time will cause the preceeding 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.

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 two-minute time span, the SysTimeSec tag increments from 0 to 59 two times. The following query returns the total count within the two-minute time span. The QualityDetail of 212 indicates that a counter rollover occurred during the query time range.

```
select DateTime, TagName, Value, Quality, QualityDetail
  as QD from History
   where TagName = 'systimesec'
      and DateTime >= '2009-08-13 1:00'
      and DateTime < '2009-08-13 1:02'
      and wwRetrievalMode = 'counter'
      and wwCycleCount = 1
```

The results are:

DateTime	TagName	Value	Quality	QD
2009-08-13 01:00:00.0000000	SysTimeSec	120	0	212

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.

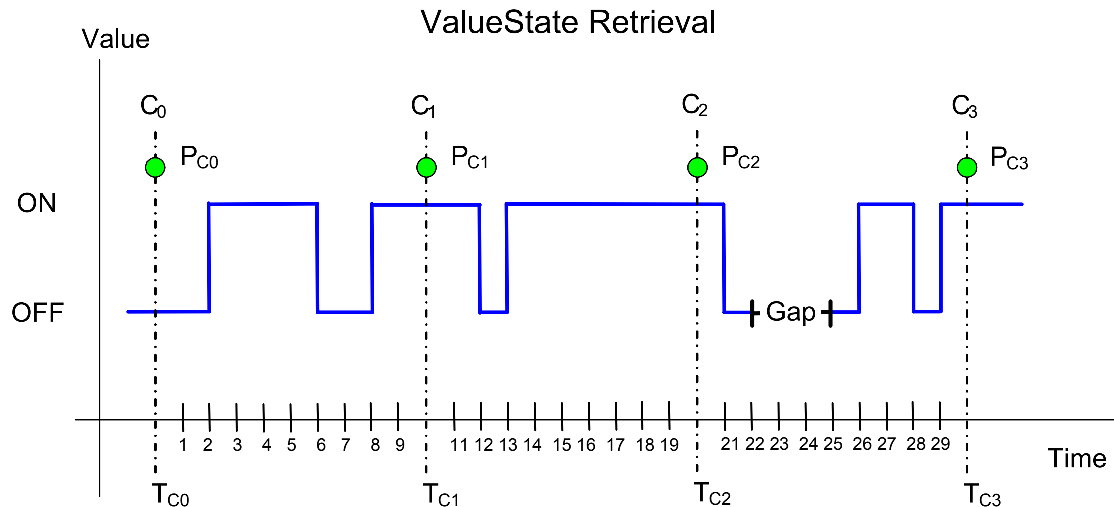
ValueState retrieval works with integer, discrete, string, and state summary tags. For other types of tags, no rows are returned. NULL values are treated like any other distinct state.

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 252.

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 preceeds 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 occcured 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 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 "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 occurrances 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 219
- Resolution (Values Spaced Every X ms) (wwResolution) on page 222
- History Version (wwVersion) on page 235
- Timestamp Rule (wwTimestampRule) on page 240
- Quality Rule (wwQualityRule) on page 244
- State Calculation (wwStateCalc) on page 252

ValueState Retrieval - Query Examples

To use theValueState 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.

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

Query 2: Minimum Time in State for a Single Tag

The following query finds the minimum time-in-state for the SteamValve 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 ('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'
      AND State = '1'
```

The results are:

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

Query 2

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'
```

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

Query 3

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
2005-04-14 00:00:00	Off	29020872.0	TOTAL

Query 4

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

Query 5: 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 rows 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.

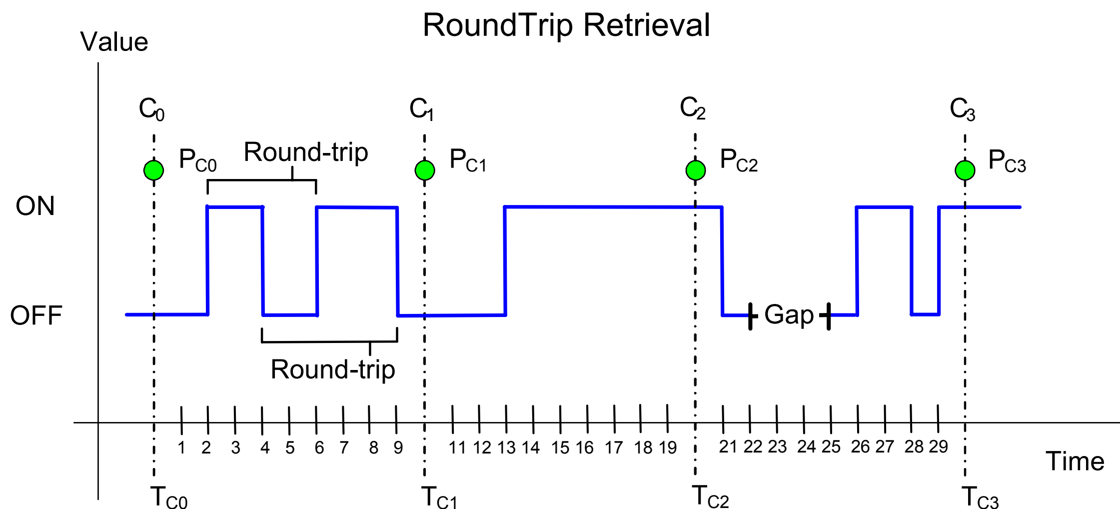
RoundTrip retrieval is supported for the History and StateWideHistory tables.

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 240
- Quality Rule (wwQualityRule) on page 244
- State Calculation (wwStateCalc) on page 252

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
```

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.

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 217.

•

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)	Resolution (Values Spaced Every X ms) (wwResolution)	Time Deadband (wwTimeDeadband)	Value Deadband (wwValueDeadband)	History Version (wwVersion)	Interpolation Type (wwInterpolationType)	Timestamp Rule (wwTimestampRule)	Quality Rule (wwQualityRule)	State Calculation (wwStateCalc)
Cyclic Retrieval	•	•			•		•*		
Delta Retrieval			•	•	•				
Full Retrieval					•				
Interpolated Retrieval	•	•			•	•	•	•	
"Best Fit" Retrieval	•	•			•	•		•	
Average Retrieval	•	•			•	•	•	•	
Minimum Retrieval	•	•			•			•	
Maximum Retrieval	•	•			•			•	
Integral Retrieval	•	•			•	•	•	•	
Slope Retrieval					•			•	
Counter Retrieval	•	•			•		•	•	
ValueState Retrieval	•	•			•		•	•	•

	Cycle Count (X Values over Equal Time Intervals)	Resolution (Values Spaced Every X ms) (wwResolution)	Time Deadband (wwTimeDeadband)	Value Deadband (wwValueDeadband)	History Version (wwVersion)	Interpolation Type (wwInterpolationType)	Timestamp Rule (wwTimestampRule)	Quality Rule (wwQualityRule)	State Calculation (wwStateCalc)
RoundTrip Retrieval	•	•			•		•	•	•

* (only on Wonderware Historian 9.0 and later)

Using Retrieval Options in a Transact-SQL Statement

You can retrieve data in the Wonderware Historian extension tables using normal Transact-SQL code, as well as the specialized SQL time domain extensions provided by the Wonderware Historian. The Wonderware 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 Wonderware Historian OLE DB Provider Unsupported Syntax and Limitations on page 137.

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).

This option is relevant in the following retrieval modes:

- Cyclic Retrieval
- Interpolated Retrieval
- “Best Fit” Retrieval
- Average Retrieval
- Minimum Retrieval
- Maximum Retrieval
- Integral Retrieval
- Counter Retrieval
- ValueState Retrieval
- RoundTrip Retrieval

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

The following queries demonstrate the cycle count behavior if applied to a single tag or to multiple tags in the same query.

Query 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

Query 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 219.

This option is relevant in the following retrieval modes:

- Cyclic Retrieval
- Interpolated Retrieval
- "Best Fit" Retrieval
- Average Retrieval
- Minimum Retrieval
- Maximum Retrieval
- Integral Retrieval
- Counter Retrieval
- ValueState Retrieval
- RoundTrip Retrieval

For delta retrieval, you can achieve similar results by using a time deadband. For more information, see Time Deadband (wwTimeDeadband) on page 227.

Resolution - Query Examples

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

Simple use of phantom cycle	Cycles not defined, but similar simple use of time before query start time	Phantom cycle used to calculate aggregates
		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 >= '2009-10-16 08:00:00'
         AND DateTime < '2009-10-16 09:00:00'
         AND wwRetrievalMode = 'Avg'
         AND wwResolution = 3600000
```

The results are:

DateTime	Value	Quality	QualityDetail	OPCQuality
2009-10-16 08:00:00.0000000	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 actually not required because the resolution has been 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 as QD,
  OPCQuality
   FROM History
      WHERE TagName IN ('SysTimeSec')
         AND DateTime >= '2007-12-11 08:00:00'
         AND DateTime <= '2007-12-11 09:00:00'
         AND wwRetrievalMode = 'Avg'
         AND wwCycleCount = 1
```

The results are:

DateTime	Value	Quality	QD	OPCQuality
2009-10-16 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.

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
- Interpolated
- BestFit
- Min

- Max
- Average
- Integral
- Counter
- ValueState
- RoundTrip

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 316.

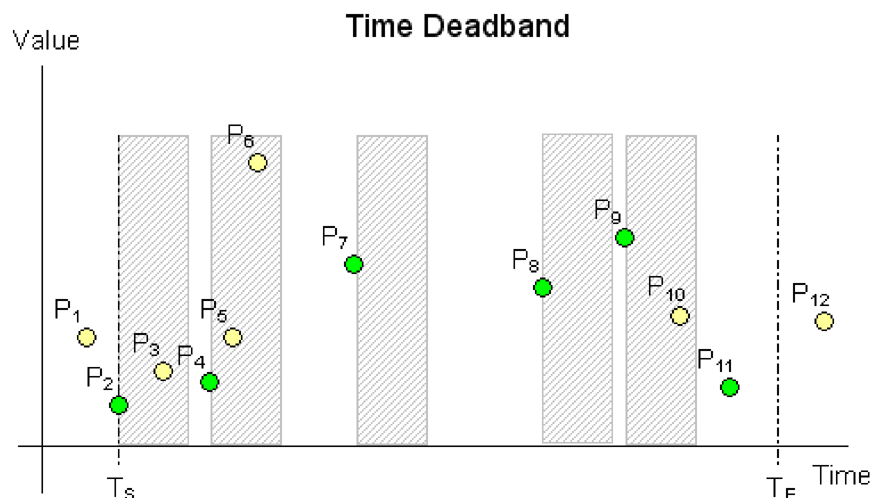
Time Deadband (wwTimeDeadband)

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 grey 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.

The following queries return data values for the analog tag 'SysTimeSec'.

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

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

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

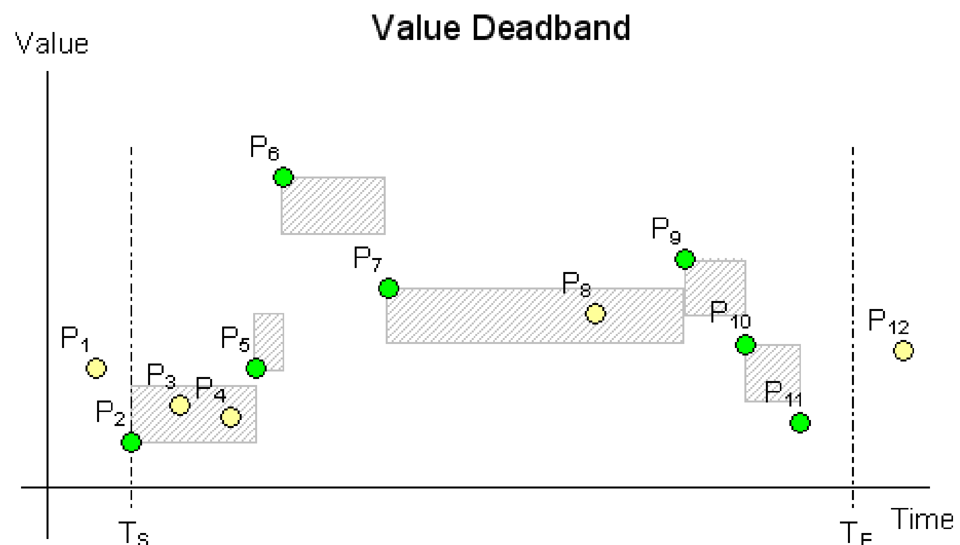
Value Deadband (wwValueDeadband)

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 grey 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

The following queries return data values for the analog tag 'SysTimeSec'. The minimum engineering unit for 'SysTimeSec' is 0, and the maximum engineering unit is 59.

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

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 Wonderware 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

For example:

```
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'
```

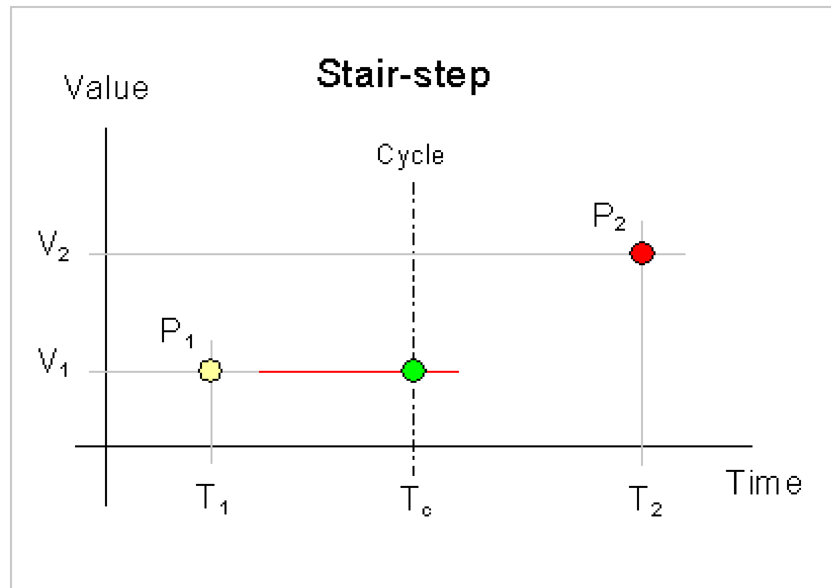
The results are:

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:

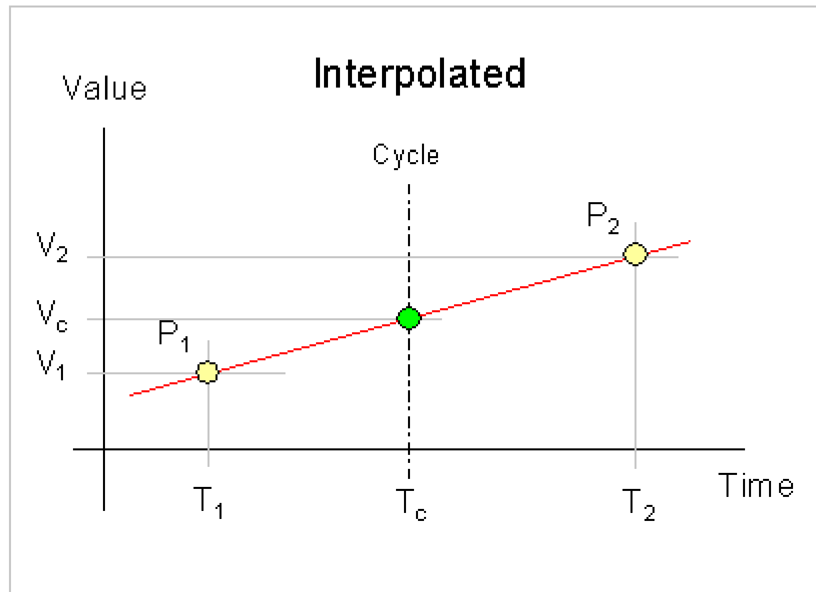
- **Stairstep:** No interpolation occurs. The value at the cycle boundary is assumed to be the same value as the last stored value before the cycle boundary.The last known point is returned with the given cycle time. If no valid value can be found, a NULL is returned.



- **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:

- Interpolated Retrieval
- "Best Fit" Retrieval
- Average Retrieval
- Integral Retrieval

The quality of an interpolated point is determined by the wwQualityRule setting. For more information, see Quality Rule (wwQualityRule) on page 244.

The interpolation type can be set on three levels:

- The Wonderware Historian system-wide setting. The system-wide setting must be either stair-step or interpolated. For more information, see System Parameters on page 33. This setting is configured using the Wonderware 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 Wonderware 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:

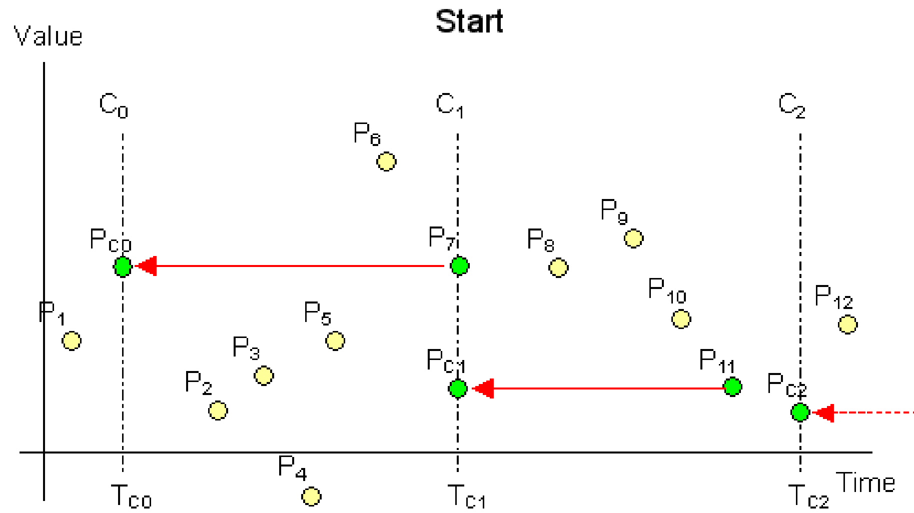
```
AND wwTimeStampRule = 'End'
```

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:

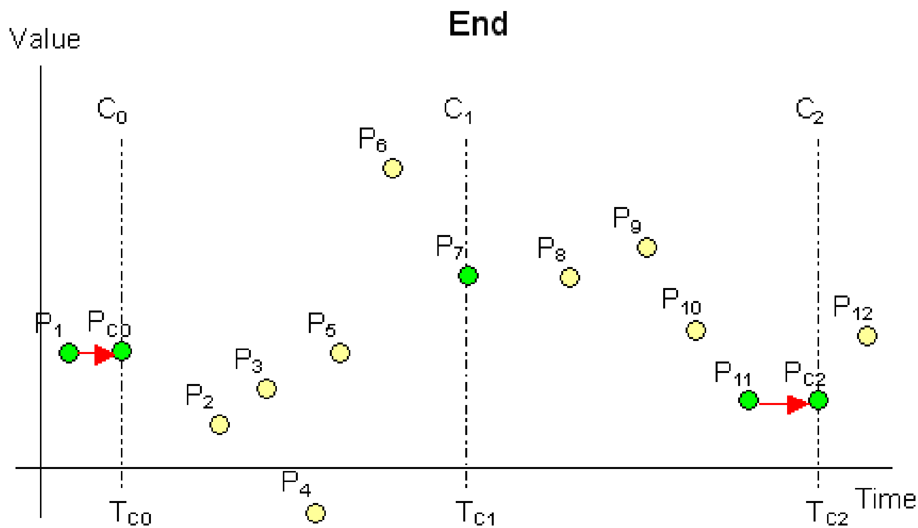
- **Start:** The value for a given cycle is stamped with the cycle start time. For example, in the following illustration of a cyclic query, the following values are returned at the cycle boundaries:
    - At T_{C0}: P_7, because it falls on the cycle boundary. In cyclic mode, if there is a value right 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}



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

This option is relevant in the following retrieval modes:

- Cyclic Retrieval (only for Wonderware Historian 9.0 and later)
- Interpolated Retrieval
- Average Retrieval
- Integral Retrieval
- Counter Retrieval
- ValueState Retrieval
- RoundTrip Retrieval

Time Zone (wwTimeZone)

For Wonderware Historian version 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. In addition to specifying the name of the timezone in the wwTimeZone parameter, you can also specify the TimeZoneID (as a string). For example, on a typical US English system, specifying "`wwTimeZone = 'Mountain Standard Time'`" and "`wwTimeZone = '64'`" yields the same result.

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 sub-system 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 Wonderware 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

If you are using date/time functions and the wwTimeZone parameter, you will need to use the **faaTZgetdate()** function.

Quality Rule (wwQualityRule)

For various retrieval modes, you can explicitly exclude values with uncertain quality from data retrieval in modes that calculate return values.

Where applicable, 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.

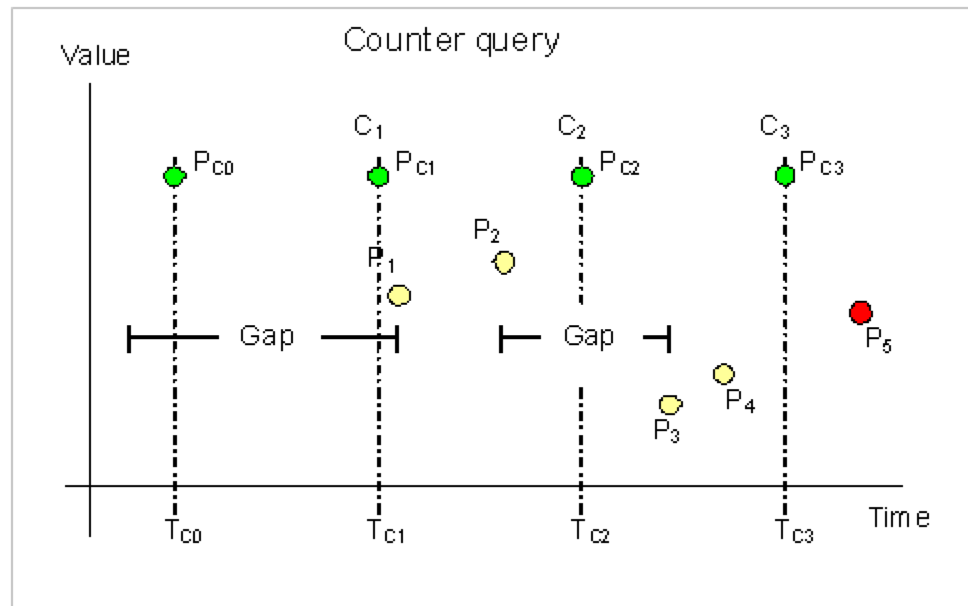
- A quality rule of GOOD means that data values with uncertain (64) OPC quality are not used in the retrieval calculations and are ignored. Values with bad QualityDetail indicate gaps in the data.
- A quality rule of EXTENDED means that data values with both good and uncertain OPC quality are used in 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 in the counter and integral retrieval modes 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 in both retrieval modes. 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 194.

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, P_1 and P_2. The query uses P_2 as the end value of the cycle and infers a start value of the cycle from P_1. At the third cycle boundary, T_{c2}, the query returns $P_2 – P_1$. Similarly, at the last cycle boundary, the query returns $P_4 – P_3$.

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'`

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
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

Similarly, 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

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	PercentGood	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

```
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
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

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
F0R5	2009-09-12 00:30	0.820	64	64	AVERAGE
F0R5	2009-09-12 00:40	1.751	64	64	AVERAGE

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

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 role over 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

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
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
I0R5	2009-09-12 00:20	0.8	192	192
I0R5	2009-09-12 00:30	2.3	192	192
I0R5	2009-09-12 00:40	1.0	192	219

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.

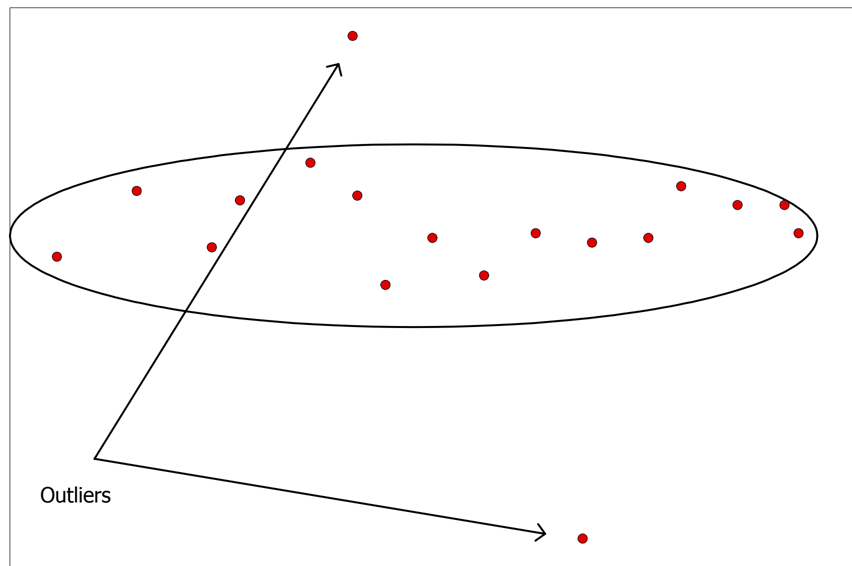
The default value for the wwFilter column is 'NoFilter'. If the query does not specify the wwFilter element at all, or if its default value is not overridden, then no filter is applied.

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, μ (mu), and time weighted standard deviation, σ (sigma), are found for each analog tag for the entire query range including phantom cycles if any, and points falling outside of the range [μ - nσ, μ + nσ] are removed from the point set before the points are processed further. In other words, the value will be filtered out if value > μ + nσ or value < μ – nσ.

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:

$$s^{2}_{\text{weighted}} = \underset{i=1}{\overset{N}{?}} w_i (x_i - \mu *)^2$$

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 264.

"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(0.1, 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 occurrance within the summary period.
LAST	The last value that occurs within the summary period.	The actual timestamp of the last value occurrance 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 occurrance 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 occurrance within the summary period.
AVG or AVERAGE	A time-weighted average calculated from values within the summary period.	The summary period start time.
INTEGRAL	An integral value calculated from values within the summary period.	The summary period start time.

Value Selector Setting	Value Returned	Timestamp Returned
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 occurrance within the summary period is used.
Delta	The last value within the summary period is used. The actual timestamp of the last value occurrance within the summary period is used.
Full	The last value within the summary period is used. The actual timestamp of the last value occurrance 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 occurrance within the summary period is used.

Retrieval Mode	Analog Summary Behavior
Maximum	The first maximum value within the summary period is used. The actual timestamp of the first maximum value occurrance within the summary period is used.
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 occurrance 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.

Edge Detection for Events (wwEdgeDetection)

An event is the moment at which a detection criterion is met on historical tag values in the Wonderware 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.

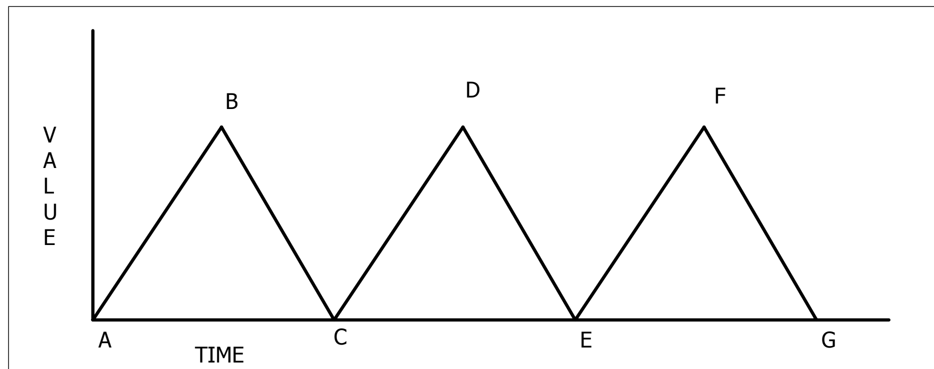
Edge detection only applies to analog and discrete value detectors. Also, edge detection is handled slightly differently based on whether you are using analog tags or discrete tags.

For more information on the event system, see Chapter 10, Event Subsystem.

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 257.

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:

	=	<	>	<=	>=
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 265. 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 265. 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

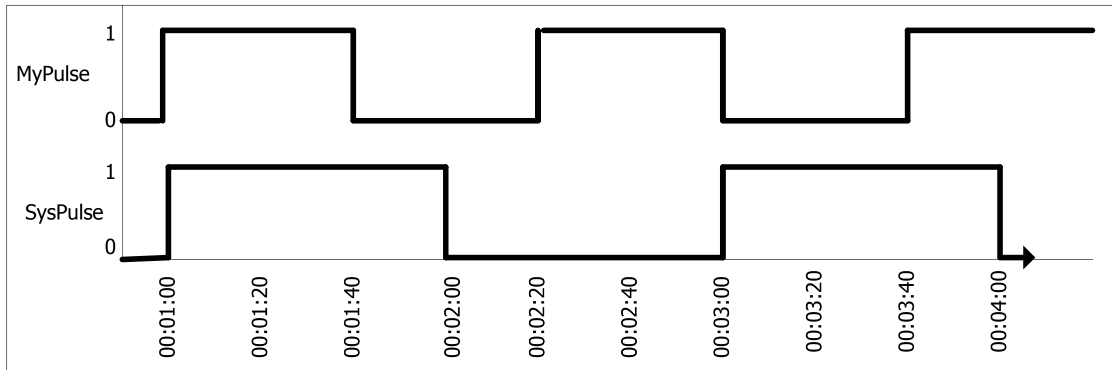
Compare these results with the same query using no edge detection, as shown in Edge Detection for Analog Tags on page 265. Notice that value of the first row in the original query is returned in the result set.

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.

Query 1

Query for *History*.

```
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

Query 2

Query for *WideHistory*.

```
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.

Query 1

For a query on 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

Query 2

For a query on 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 268. 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).

Query 1

For a query on the *History* table, if the WHERE clause criteria specifies to return 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

Query 2

For a query on the *WideHistory* table, applying 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 268. 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.

Query 1

```
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

Query 2

```
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 268.

Chapter 8

Query Examples

In addition to query examples that use the Wonderware 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" in Chapter 1, "Table Categories," in your *Wonderware 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 data for each tag in the table. For more information, see "History Tables" in Chapter 1, "Table Categories," in your *Wonderware 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" in Chapter 1, "Table Categories," in your *Wonderware 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	SysTimeMin	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 withing a single query.

The following query returns the minimum, maximum, and average values for the SysTimeSec tag for the last minute.

```
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
```

Querying the StateSummaryHistory View

The StateSummaryHistory view is a “wide” view that allows you to return multiple statistics for a single tag withing 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 minute. One row is returned for each state.

```
declare @End datetime

set @End =
  left(convert(varchar(30),getdate(),120),14)+'00:00'

SELECT TagName, Value, OPCQuality, StateCount,
  StateTimeTotal, StateTimePercent
   FROM StateSummaryHistory
      WHERE TagName = 'SysPulse'
         AND StartDateTime >= dateadd(minute,-60,@End)
         AND EndDateTime < @End
         AND wwCycleCount = 2
```

The results are:

TagName	Value	OPCQuality	StateCount	StateTimeTotal	StateTimePercent
SysPulse	0	192	15	900000	50
SysPulse	1	192	15	900000	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
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

TagName	StartDateTime	EndDateTime	STM	STMC	Value
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 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 "Naming Conventions for Tags" on page 23.

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>

Query 1

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'
```

Query 2

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 only satisfied 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

```
DateTime                    TagName      Value
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
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. *N* = 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 Wonderware 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 tag 'SysTimeSec' 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

.
.
.

If you want to use date/time functions and the wwTimeZone parameter in the same query, you will need to use the **faaTZgetdate()** function. This is because of differences in how the SQL Server and the Wonderware Historian OLE DB provider determine the end date for a query.

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 Wonderware Historian OLE DB provider. The Wonderware 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 Wonderware 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.

To work around this problem, use the **faaTZgetdate()** function with intermediate variables. For example:

```
DECLARE @starttime datetime

SET @starttime = dbo.faaTZgetdate('eastern daylight
  time')

SELECT DateTime, TagName, Value FROM History
   WHERE TagName IN ('SysTimeHour', 'SysTimeMin',
   'SysTimeSec')
      AND DateTime > DateAdd(mi, -30, @starttime)
      AND DateTime < DateAdd(mi, -5, @starttime)
      AND wwTimeZone = 'eastern daylight time'
```

The following example uses a wide table:

```
SELECT * FROM OpenQuery(INSQL, '
    SELECT DateTime, SysTimeHour, SysTimeMin,
    SysTimeSec FROM WideHistory
    WHERE DateTime > DateAdd(mi, -30,
    faaTZgetdate("eastern daylight time"))
        AND DateTime < DateAdd(mi, -5,
        faaTZgetdate("eastern daylight time"))
        AND wwTimeZone = "eastern daylight time"
    ')
```

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.

Using Two Equality 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)
```

Using One Equality 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)
```

Using No Equality 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

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

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

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
2001-01-13 12:01:00.000     0

(60 row(s) affected)
```

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 Wonderware 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:

	EventKey	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	EventKey	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 Wonderware Historian OLE DB Provider. Stored procedures can use any valid Transact-SQL syntax to access Wonderware 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.

Querying Data to a Millisecond Resolution using SQL Server 2005

Internally, the Wonderware Historian uses a Win32 **FILETIME** data type for representing date/time. This 64-bit integer yields a time resolution of 100 nano-seconds. From a retrieval point of view, however, the limiting factor is the resolution of the SQL Server 2005 **datetime** type. For SQL Server 2005, datetime represents time with a resolution of 1/300 second, or 3.33 milliseconds. SQL Server 2005 rounds the time values in increments of .000, .003, or .007 seconds.

Note When you are using Wonderware Historian with SQL Server 2008, millisecond resolution is supported for retrieval by using the DateTime2 data type.

For consistency with SQL Server 2005, it is important that any OLE DB time functions present time data according to the same convention. A rounding algorithm is incorporated to achieve the desired results. This is necessary when a user calls a datetime function that manipulates milliseconds within an OPENQUERY statement. For example:

```
SELECT * FROM OpenQuery(INSQL,
      'SELECT DATEPART(millisecond, getdate()),
      Value FROM Live WHERE TagName =
      "ReactTemp"')
```

returns results in the same format as:

```
SELECT DATEPART(millisecond, getdate())
```

If you require millisecond time resolution, the Wonderware Historian provides a means of bypassing the SQL Server 2005 datetime restriction by retrieving the datetime as a string. This is implemented by the use of specific style specifiers for the CONVERT function. These specifiers (numbered 909, 913, 914, 921, and 926) return a string in the same format as the corresponding 100-series style specifiers, but with millisecond time resolution. Style specifier 909 formats the string in the same way as standard specifier 109; 913 formats as for 113; and so on.

For details of the string format corresponding to each 100-series specifier, see the description of the CONVERT(data_type, expression, style) function in your Microsoft SQL Server 2005 documentation.

An OPENQUERY statement must be used to ensure that the CONVERT statement is passed to the Wonderware Historian parser, because this is a specific extension to CONVERT functionality provided by the Wonderware Historian.

For example:

```
SELECT CONVERT(varchar(30), DateTime, 113), Value
   FROM Live
      WHERE TagName = 'ReactTemp'
```

might return '2001-08-10 14:30:45:337' (3.33 millisecond resolution), while

```
SELECT * FROM OpenQuery(INSQL,
   'SELECT CONVERT(varchar(30), DateTime, 913), Value
   FROM Live
      WHERE TagName = "ReactTemp" ')
```

would return '2001-08-10 14:30:45:**335**' (1 millisecond resolution).

In all, a total of ten CONVERT styles are available within the Wonderware Historian. These are 20, 120, 21, 121, 126, 909, 913, 914, 921, and 926.

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

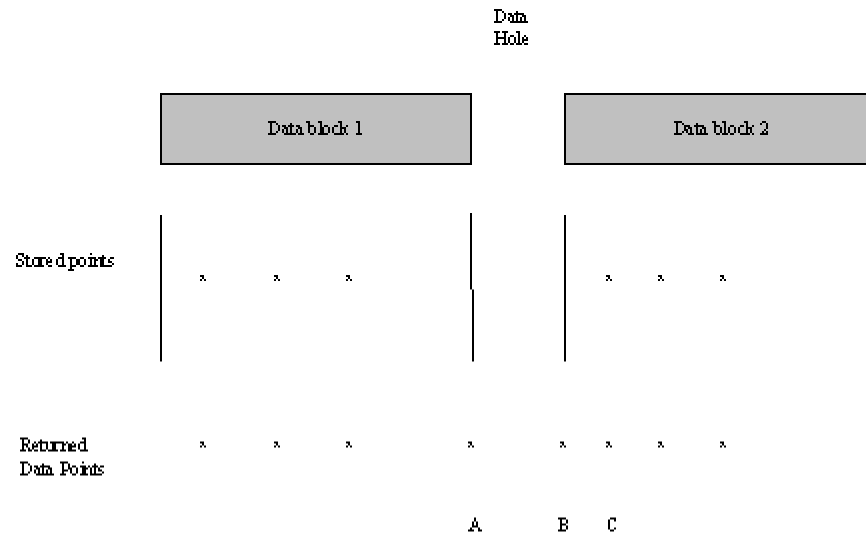
You cannot use variables in an OPENQUERY statement. Therefore, if you want to use variables in a query on the wide table, you must first build up the OPENQUERY statement "on the fly" as a string, and then execute it.

```
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
```

Retrieving Data Across a Data "Hole"

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 has been stopped between history blocks, there will be a "hole" 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

	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:

	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 hole, 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:

	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 Wonderware 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.

Retrieving Data from History Blocks and the Active Image

During initialization, the retrieval subsystem will note the oldest timestamp for each tag in the active image. If the query start time is more recent than the oldest timestamp, all of the requested tag values will be obtained directly from the active image. If the query start time is older than the oldest timestamp, data will be retrieved from the history blocks.

If the query spans up to the oldest timestamp, the retrieval subsystem will check the active image to re-determine oldest timestamp (because older values in the active image are overwritten to make room for new values). If the requested data is still in the active image, it will be used. Otherwise, data is retrieved from the history blocks. This process will continue until the active image can be used or the query end time is surpassed. It is important to note that this process is applied on a per-tag basis.

For more information, see "How the Active Image Storage Option Affects Data Retrieval" on page 121.

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 224.

```
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:

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

Chapter 9

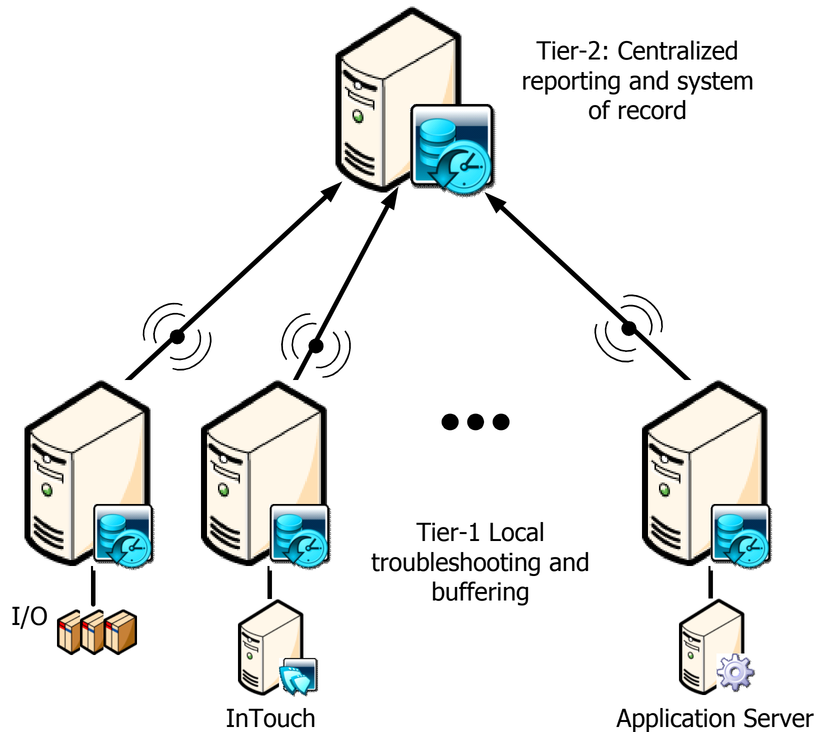
Replication Subsystem

Data from one Wonderware Historian can be replicated to one or more other Wonderware Historians, creating a "tiered" relationship between the historians.

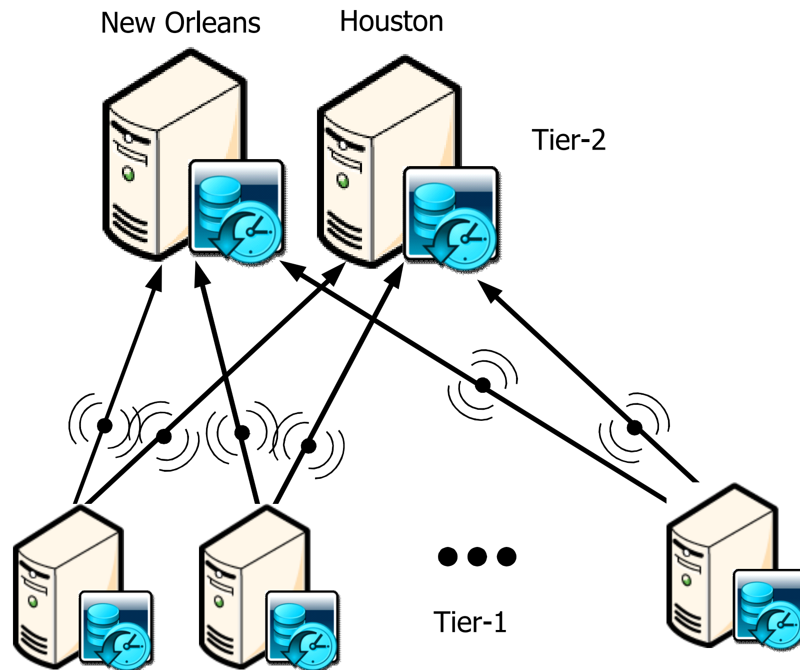
About Tiered Historians

You can set up Wonderware Historians in a variety of tiered configurations. In a common configuration, data from multiple individual historians (called tier-1 historians) is fed into a single centralized historian (called a tier-2 historian).

The tier-2 historian stores data replicated from the tier-1 historians.



Another configuration is to have multiple tier-1 historians that feed information to multiple tier-2 historians in a many-to-many relationship.



How Tags are Used During Replication

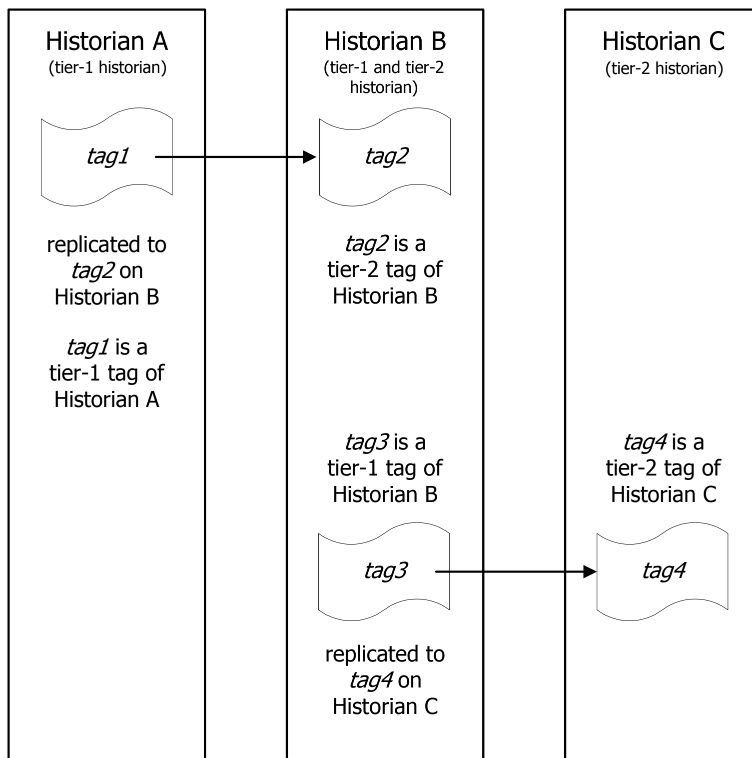
Data from a tier-1 historian is replicated to a tier-2 historian using tags in the same way that information is collected by an individual Wonderware historian.

The tier for a tag is determined by where it comes from:

- Values for tier-1 tags are gathered directly from an IDAS or MDAS.
- Values for tier-2 tags come from another Wonderware Historian server.

A historian can act as a tier-1 and a tier-2 historian simultaneously.

A typical scenario for a tiered historian appears in the following example. Tag1 is collected on historian A and all its values are replicated to historian B, where they are stored as values of tag2. At the same time historian B collects data for its tag3 and all its values are replicated to historian C, where they are stored as values of tag4.



In this example, the tags are identified as follows:

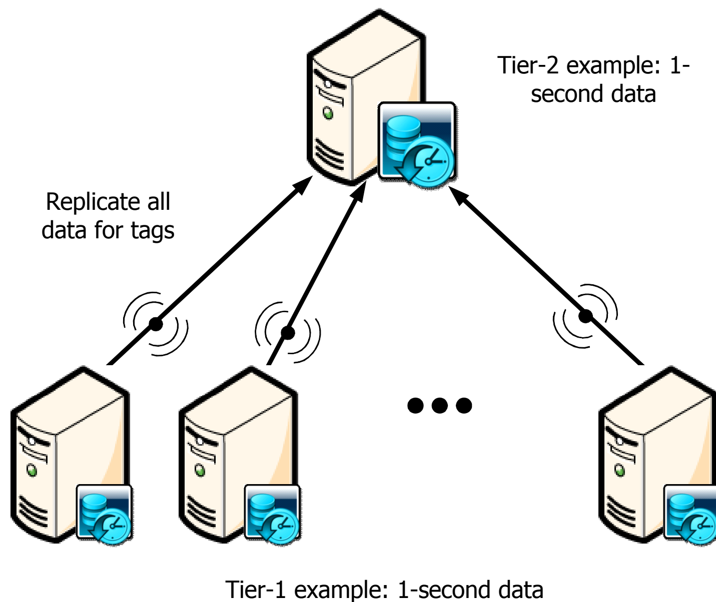
- tag1 is a tier-1 tag of historian A
- tag2 is a tier-2 tag of historian B
- tag3 is a tier-1 tag of historian B
- tag4 is a tier-2 tag of historian C
- historian A is a tier-1 historian
- historian B is both tier-1 and tier-2 historian
- historian C is a tier-2 historian

Important Be careful not to create or modify a replicated tag on a tier-1 historian to have the same tagname that already exists on a tier-2 historian. The system does not prevent you from having a replicated tag on a tier-2 historian receiving data from two or more different tier-1 historians. However, when you retrieve data for that replicated tag on the tier-2 historian using the tagname, an incorrect blend of data from the two (or more) data sources is returned.

There are two types of replication: simple replication and summary replication. Summary replication provides periodic summaries of high resolution data, while simple replication retains the original data resolution.

Simple Replication

When a tag is configured for simple replication, all values stored in the tier 1 historian are replicated to the tier 2 server. Analog, discrete, and string tags can be configured for simple replication. Replicated tags of a tier-2 historian cannot be configured for further replication.



The results of replication are stored on the tier-2 historian using the same tag type as was used for the tag on the tier-1 historian.

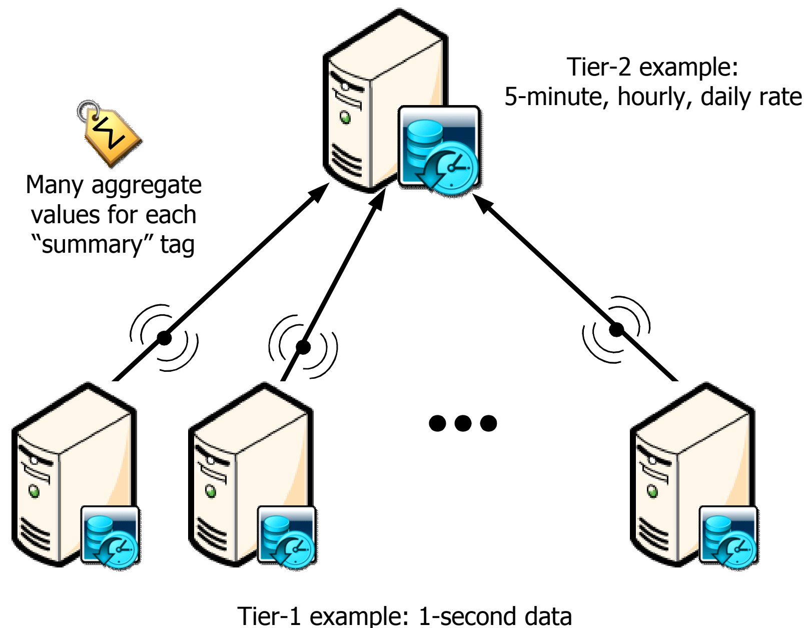
Simple replication of tag values occurs:

- Every time a new stored data value arrives into the active image of the tier-1 historian.
- Every time there is a change made in the history of the tier-1 tag to keep tier-1 and tier-2 tag values synchronized in history. For more information, see "How Replication is Handled for Different Types of Data" on page 331.

Summary Replication

Summary replication involves analyzing and storing statistical information about the tag value at the tier-1 historian. This occurs at an interval you specify, called the calculation cycle.

The result of the calculation is sent to the tier-2 historian to be stored with the timestamp of the cycle. Tier-2 tags are not dependent on the "real-time window" that usually applies to tier-1 tags.



There are two types of summary replication:

- Analog summary replication
- State summary replication

The results of summaries are stored on the tier-2 historian using analog summary or state summary tags.

If a tier-1 historian is unable to perform a scheduled summary calculation for any reason, it adds a record about the event into a replication queue. When there are enough system resources available, or there is a specific event from another subsystem, the tier-1 historian can perform the summary calculations and clear the queue.

Analog Summary Replication

Analog summary replication produces summary statistics for analog tags. The statistics relate only to the recorded interval. Statistics available are:

- Time-weighted average
- Standard deviation
- Integral
- First value in a period with timestamp
- Last value in a period with timestamp
- Minimum value in a period with timestamp
- Maximum value in a period with timestamp
- Start time of summary period
- End time of summary period
- OPC Quality
- Percentage of values with Good quality
- Value

When you retrieve the data, you specify which calculation you want to return. For more information, see "Querying the AnalogSummaryHistory View" on page 279.

The functionality provided by analog summary replication is similar to using the minimum, maximum, average, and integral retrieval modes. For a comparison example, see Querying Aggregate Data in Different Ways on page 316.

When you use the Wonderware Historian SDK to retrieve analog summary tag data, the values returned through the SDK for analog summary tags from history correspond to the "Last" values in the AnalogSummaryHistory table when using defaults. Use the corresponding retrieval mode to get the minimum, maximum, average, slope, and integral values.

State Summary Replication

State summary replication summarizes the states of a tag value. State summary replication can be applied to analog (integer only), discrete, and string tags.

You use this for analyzing process variables with a limited number of states, such as a machine's state of running/starting/stopping/off. State summary replication provides the following, for each distinct state:

- Total time
- Percent of the cycle
- Shortest time
- Longest time
- Average time
- OPC Quality
- Value

A state summary results in a series of values, each representing a different state, for the same tag and time period.

When you retrieve the data, you specify which calculation you want to return. For more information, see "Querying the StateSummaryHistory View" on page 280.

The functionality provided by analog summary replication is similar to using the ValueState and RoundTrip retrieval modes.

You can define state summary replication for a large number of states, but state data is dropped if the number of states occurring in the same reporting period exceeds the maximum number of states allowed. You configure the maximum states when you create the state summary tag. The default number of maximum states is 10. The replication subsystem will calculate summaries for the first 10 distinct states, in the order in which they occur in the data (not in numeric or alphabetic order). Be aware that the higher the number of maximum states, the more system resources are used to keep track of the time-in-state for each distinct state.

Replication Schedules

Each real-time summary has a specified schedule, by which the summary is calculated and sent to the tier-2 historian to be stored with the timestamp of the cycle.

There are two types of replication schedules:

- Periodic replication schedules

  You can configure a summary to replicate based on an cycle such as 1 minute, 5 minutes, 1 hour, 1 day, and so on. The cycle boundaries are calculated starting at midnight, tier-1 server local time, and continue in fixed time increments. The time between cycles is constant within a day even through a daylight savings change. Note that the last cycle in a day may be shorter to force replication at midnight. The calculation cycle starts at midnight. For example, a 13-minute cycle is stored at 12:00 a.m., 12:13, 12:26, ... 11:27 p.m., 11:40, 11:53 and then again at 12:00 a.m.
- Custom replication schedules

  Custom schedules force replication cycles to occur at fixed times of the day in tier-1 server local time. You choose the fixed times of day.

Replication Schedules and Daylight Savings Time

Daylight Savings Time affects replication schedules that are triggered according to a time period, such as every hour, every thirty minutes, and so on. Replication schedules that are triggered at a fixed time that you specify are not affected.

In the following examples, the time change occurs at 2:00 a.m.

In this example, the summary period is configured to be every 30 minutes. On the "fall back" day, will be two extra summaries performed during the repeated hour for that day. For the "spring forward" day, there will be two summaries missing because of the skipped hour. The next replication occurs at the next scheduled time. In this case, it would be 3:00 a.m.

Summary Period = 30 Minutes

"Fall Back" Day		Regular Day		"Spring Forward" Day	
1:00 a.m.	Daylight	1:00 a.m.	Standard	1:00 a.m.	Standard
1:30 a.m.	Daylight	1:30 a.m.	Standard	1:30 a.m.	Standard
1:00 a.m.	Standard	2:00 a.m.	Standard		
1:30 a.m.	Standard	2:30 a.m.	Standard		
2:00 a.m.	Standard	3:00 a.m.	Standard	3:00 a.m.	Daylight
2:30 a.m.	Standard	3:30 a.m.	Standard	3:30 a.m.	Daylight
3:00 a.m.	Standard				
3:30 a.m.	Standard				

extra summaries

time gap

In the next example, the summary period is configured for every four hours. The scheduled summaries do not fall exactly on or within the boundaries of the time change hour. In this case, on the "fall back" day, the summary subsequent to the time change hour includes four hours of data for the "fall back" day. An extra summary for an hour's worth of data is performed at the end of the "fall back" day. On the "spring forward" day, the summary period that contains the skipped hour includes one less hour of data.

Summary Period = 4 Hours

"Fall Back" Day		Regular Day		"Spring Forward" Day	
0:00	Daylight	0:00	Standard	0:00	Standard
3:00	Standard	4:00	Standard	5:00	Daylight
7:00	Standard	8:00	Standard	9:00	Daylight
11:00	Standard	12:00	Standard	14:00	Daylight
15:00	Standard	16:00	Standard	17:00	Daylight
19:00	Standard	20:00	Standard	21:00	Daylight
23:00	Standard	24:00	Standard	0:00	Daylight
0:00	Standard				

4 hours of data summarized

"extra" hour

4 hours of data summarized

missing an hour here

For a custom summary period, the summaries always occur at the fixed times of day that you specify in local time. However, the summary includes and extra hour of data for the "fall back" day (because of the overlap hour) and for the "spring forward" day (because of the skipped hour).

Summary Period = Custom: 0, 4, 8, 12, 16, 20

"Fall Back" Day		Regular Day		"Spring Forward" Day	
0:00	Daylight	0:00	Standard	0:00	Standard
4:00	Standard	4:00	Standard	4:00	Daylight
8:00	Standard	8:00	Standard	8:00	Daylight
12:00	Standard	12:00	Standard	12:00	Daylight
16:00	Standard	16:00	Standard	16:00	Daylight
20:00	Standard	20:00	Standard	20:00	Daylight
0:00	Standard	24:00	Standard	0:00	Daylight

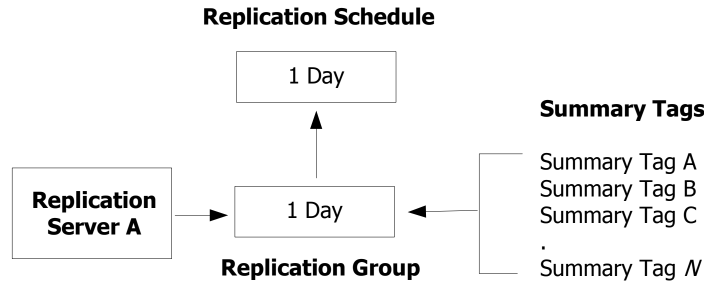
extra hour - 5 hours of data summarized

only 3 hours of data summarized

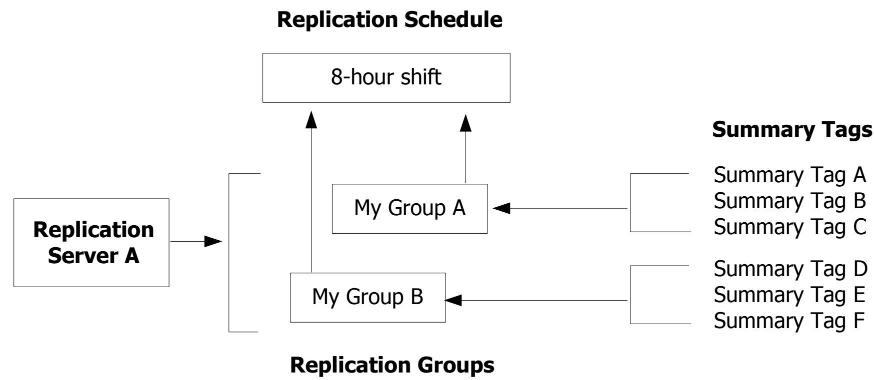
If a Daylight Savings Time change causes a scheduled time to be ambiguous, such as 1:30 a.m. on a "fall back" day when the clock jumps from 1:59 a.m. Daylight Savings Time to 1:00 a.m. standard time and the time could be interpreted as 1:30 a.m. Daylight Savings Time or 1:30 a.m. standard time, the replication will occur at the latter of the two occurrences. In this case it would be 1:30 a.m. standard time.

Replication Groups

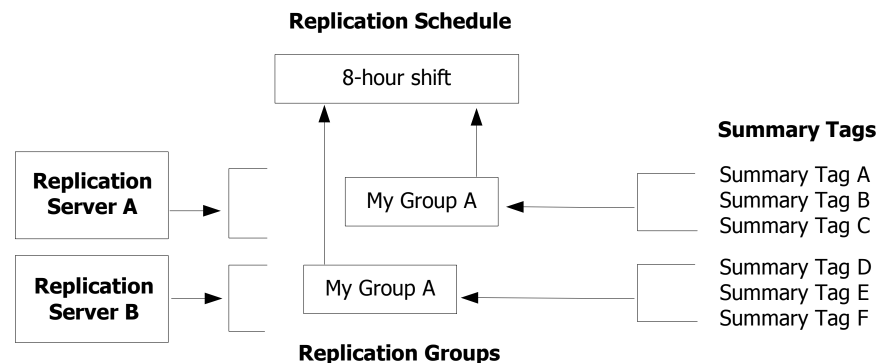
A replication group abstracts a tag from a schedule. You can assign multiple summary tags to a single replication group.



Multiple groups can be assigned to a single schedule. This simplifies maintenance. If you need to edit the schedule (for example, change the time of day for a shift end), you only need to edit the replication schedule, not the individual groups or summary tag configurations.



A replication group must be unique for a type of summary tag, either analog or state. You can, however, have the same group name for analog summary tags as you do for state summary tags. You can also have the same replication group defined in multiple servers.



How Replication is Handled for Different Types of Data

An accurate mapping between the data on the tier-1 and tier-2 historians is maintained over time. This mapping includes both tag configuration and data synchronization.

Replication is unidirectional, from tier-1 to tier-2 historians. If the data on the tier-2 historian is changed in any way, the system does not try to map the change back to the tier-1 historian.

Assume that some *tag1* of historian *A* is configured to be replicated in real-time to *tag2* of historian *B*. The *tag2* of historian *B* will have exactly the same data and OPC quality values as *tag1* of historian *A*. The replication system performs the following actions:

- When a new original value fitting the real-time window gets stored on historian *A*, it gets transmitted and stored on historian *B*, as well as the original value.
- If you perform an insert or update operation for some old values of the historian A, the same change is reflected on historian B.
- If some store-and-forward data gets merged into history on historian A, the same data gets transmitted to historian B and gets merged into history of historian B.

Replication is implemented in two ways: streaming replication and queued replication. The replication system uses a combination of streamed replication and queued replication as required.

Streaming Replication

When values of tier-1 tags are received from an IDAS or MDAS (Wonderware Application Server) and arrive at the tier-1 historian as a time-ordered data stream directly to the real-time data storage service, the historian not only stores the data, but also forwards it to the replication subsystem if replication is configured for those tags.

Then the replication subsystem immediately streams that data to the tier-2 historian for simple replication, or performs summary calculations and streams the resulting summaries.

This happens equally efficiently for tag values of timestamps close to the current system time and for late data tags, including the case when the late data delay exceeds the real-time window.

If the tier-2 historian becomes unavailable, the replication subsystem continues to stream replicated data into the local store-and-forward path. When the connection is restored, all replicated data is sent as compressed snapshots to the tier-2 historian and incorporated into history.

Streaming replication is the fastest and most efficient way of data replication, but there are some scenarios where it cannot be used and another method called queued replication is applied.

Queued Replication

Queued replication is used for scenarios in which streaming replication is not appropriate, such as the following:

- An interruption occurs in the tier-1 data stream, such as when a remote IDAS configured for store-and-forward cannot communicate with the tier-1 historian for a long period of time. When the connection is restored and the store-and-forward data finally arrives at the tier-1 historian, it may be already streaming newer data.
- An insert, update, or CSV file import operation could be performed for tier-1 tag values, so the summaries should be recalculated for that time period and re-replicated to the tier-2 historian.
- If the tier-1 historian is started or stopped, and there are some summaries spanning across the startup/shutdown time that must be recalculated and re-replicated to the tier-2 historian.

In such cases, the replication subsystem receives notifications from the manual data storage service that contain information about what kind of tier-1 tag data (original or latest) has changed for a particular time interval. Then the replication subsystem places that notification record into the replication queue stored in the Runtime database of the tier-1 historian. Later, when the connection to the tier-2 historian is restored, the replication subsystem processes that queue by querying the tier-1 data and replicating it to the tier-2 historian. As soon as a queue item is successfully processed, it is removed from the replication queue.

Although the replication subsystem optimizes the queue by combining adjacent records, queued replication is slower and requires more system resources as compared to streamed replication, because it involves querying tier-1 data already stored on disk.

Tag Configuration Synchronization between Tiered Historians

If a summary tag gets deleted on the tier-1 historian, its corresponding tag on the tier-2 historian remains intact to allow for retrieval of data already collected. A tier-1 tag cannot be deleted from the tier-1 historian if it is being replicated--first delete the tag replication and then delete the tier-1 tag. A tier-2 tag can be deleted from the tier-2 historian, but it should only be deleted after the corresponding replication has been deleted from the tier-1 source historian. Otherwise, it will be recreated.

Replication Components

The components of the replication subsystem are:

Component	Description
Realtime Data Storage Service (aahStoreSvc.exe)	Internal process that stores real-time data to disk. This process runs as a Windows service. This process detects changes to the subscribed tags and passes those values to the Replication Service for further processing.
Manual Data Storage Service (aahManStSvc.exe)	Internal process that processes non-real-time data and stores it to disk. This process runs as a Windows service. This process is also called "alternate" storage. This process is responsible for notifying the Replication Service of inserts, updates and store-and-forward operations for the subscribed tags.
Active Image	Memory segment that temporarily hold all real-time data while the storage subsystem stores the actual values to disk.
History Blocks	Set of folders and files on disk that contain historical data.
Replication Service aahReplicationSvc.exe	Performs the data summaries on the tier-1 historian and sends the results to the tier-2 historians by means of the MDAS Server.This process runs as a Windows service.
Configuration Service (aahCfgSvc.exe)	Internal process that handles all status and configuration information throughout the system. This process runs as a Windows service. Used to configure the replication subsystem. Detects changes in the replication and summary configuration in the Runtime database and automatically reconfigures the Replication Service.
Runtime database	SQL Server database that stores all configuration information.
System Management Console client application	Tool you use to configure tier-1 and tier-2 servers, tags for replication, replication schedules, and so on.
MDAS Server (aahMDASServerSvc.exe)	At the tier-2 historian, the MDAS Server accepts incoming data from the tier-1 historian and sends it to storage.

Component	Description
Tier2Storage Engine (aahStorageEngine.exe)	Internal process performing tier-2 data storage and low-level retrieval. On the tier-2 historian, this engine receives tier-2 data from the tier-1 historian through the MDAS Server and stores it to disk in the history blocks. On the tier-1 historian, this engine accepts tier-2 data from the replication service and stores it in the store-and-forward folders if the tier-2 historians are temporarily unavailable. Several processes of this name can be running at the same time: one for the main tier-2 data storage and low-level retrieval, and the others for the store-and-forward operation (one instance per tier-2 historian).

For a complete diagram of the Wonderware Historian architecture, see "Wonderware Historian Subsystems" on page 19.

Replication Run-time Operations

When a tier-1 historian cannot communicate with a tier-2 historian because of a network outage or other reason, the replicated tags can still be configured and the data collected. When the tier-2 historian becomes available, the replication configuration and data are sent to the tier-2 historian.

System and data integrity is not guaranteed if a disorderly shutdown occurs, such as a power outage.

Replication for tags will stop if:

- You delete the source tag configuration on the tier-1 Historian.
- You configure the tier-1 tag so that its data values are not stored.
- An integer analog tag is being replicated as a state summary, and you change the source tag to be a real analog tag.

Replication Latency

Replication latency is the time it takes for the system to make a value available for retrieval on tier-2 from the moment it was stored or calculated on tier-1.

Replication latency depends primarily on whether the streaming or queued replication method is being applied in each particular case and the available system resources to execute that method in each particular case.

Streaming replication tends to have a shorter latency period than queued replication as it deals with smaller amounts of data and is processed at a higher priority.

Replication Delay for “Old” Data

The replication delay identifies how frequently “old” data, which includes inserts, updates, and store-and-forward data, is sent from the tier1 historian to the tier-2 historian. The replication delay applies only to queued replication.

You specify the delay using the OldDataSynchronizationDelay system parameter. For more information, see "System Parameters" on page 33.

This delay represents your intent, while the replication latency identifies the real time difference. If the latency period becomes longer than the replication delay, the system will not be able to maintain the expected replication.

If you set the OldDataSynchronizationDelay system parameter to zero, all changes detected in the tier-1 are immediately sent to the tier-2 , which may be very inefficient for certain applications.

Continuous Operation

If a tier-2 historian becomes unavailable and is configured for store-and-forward operations, you can still add, modify and delete replication and summary objects in the local configuration of tier-1, and store data locally for tier-2 tags created before tier-2 became unavailable or while it is still unavailable.

After the tier-2 historian is available, the Replication Service compares the latest replication and summary objects with the tier-2 tags currently existing on the tier-2 historian and performs a dynamic reconfiguration to ensure all data is synchronized. The reconfiguration history that was stored locally is also sent to the tier-2 historian and merged into its history. It will appear as though the disconnection between the tiers never took place.

Overflow Protection

If the tier-2 historian is unable to handle the incoming data from the tier-1 historian, the Replication Service detects the situation and switches into store-and-forward mode, where the data gets accumulated locally until the storage limit is reached. If the limit is reached, all data to be sent to the tier-2 historian gets discarded and an error message is logged.

Security for Data Replication

Connections from a tier-1 historian to tier-2 historian have to be authenticated before any replication task can be performed on the tier-2 historian.

A local Windows user group called aaReplicationUsers is created on the tier-2 historian during the tier-2 historian installation. The ArchestrA user account is automatically added to this group. Only members of the aaReplicationUsers group are allowed to perform replication tasks including adding, modifying, and sending values for replication tags. This group is not allowed to perform other non-replication tasks, such as adding or modifying a tier-1 tag.

When you configure a replication server on the tier-1 historian, you must specify a valid Windows user account on the tier-2 historian for the replication service to use. This security account does not have to be a valid account on the tier-1 historian or even be in the same security domain as the tier-1 historian. If no replication user credentials are configured in at the tier-1 historian, the ArchestrA user account credential is passed to the tier-2 historian for authentication.

Using Summary Replication instead of Event-Based Summaries

You can use summary replication instead of event-based summaries. Compared to event-based summaries, replicated summaries:

- Support more types of calculations.
- Are stored to history blocks, instead of in SQL Server tables, thus taking advantage of all that the Wonderware Historian storage subsystem has to offer in terms of data compression and throughput as compared to regular SQL Server database storage.
- Can be retrieved using the full array of real-time query extensions provided by the Wonderware Historian.

For more information on event-based summaries, see Chapter 10, "Event Subsystem."

Chapter 10

Event Subsystem

Plant events range from startups and shutdowns, through trips and shift changes, to batch events and operator actions.

You can use the Wonderware Historian event subsystem to detect events and associate actions when they are detected. At a basic level, anything that can be determined by examining stored data can be used as an event. The event subsystem can be configured to periodically check to see if an event occurred. This is called event detection. A subsequent action can then be triggered after an event is detected. However, there is no guarantee of immediacy for actions; in fact, other mechanisms can preempt actions under certain circumstances.

For the historian, event storage encapsulates more than just the fact that something happened. An event is the set of attributes describing the moment a detection criterion is met on historical tag values in the historian. Attributes of an event include the date and time that the event occurred in history and the date and time that it was detected. Records of detected events can be logged to the database regardless of whether or not any configured actions are subsequently initiated. In other words, sometimes it may be desirable to simply log the fact that an event occurred without initiating an action. The opposite may be true, as well.

In short, the event subsystem performs the following basic functions:

- Detects when events occur by comparing sets of criteria against historical data in the database.
- Optionally logs event records to a dedicated SQL server table (EventHistory).
- Optionally triggers a configured action each time an event is occurs.

For information about configuring events, see Chapter 11, "Configuring Events," in your *Wonderware Historian Administration Guide*.

The event subsystem does not support Daylight Savings Time changes. The replication subsystem, however, does handle Daylight Savings Time changes, and you can use replication to generate data summaries according to a schedule. For more information, see Chapter 9, "Replication Subsystem."

Event Subsystem Components

The following table describes the components of the event subsystem.

Component	Description
Configuration Editor	Part of the System Management Console. Used to specify event definitions and possible actions.
Runtime database	Stores event definition information and all data generated by the event subsystem, such as records of event detections, data summaries, and data snapshots.
Event System Service (aahEventSvc.exe)	Internal process that coordinates event detection and action functions. This process runs as a Windows service. Using the System Management Console, you can configure the event service to automatically start and stop at the same time as the Wonderware Historian. The event service is responsible for: • Reading event definition information from the Runtime database. • Creating event detectors and actions, including allocating the necessary processing threads and establishing database connections. • Initiating the event detection cycle.

Component	Description
SQL variables	Available for use in event queries.

You use the System Management Console to configure the event subsystem.

For a complete diagram of the historian architecture, see "Wonderware Historian Subsystems" on page 19.

Uses for the Event Subsystem

Generally, you should use the Wonderware Historian event subsystem to monitor non-critical system conditions that occur only occasionally. For example, possible event detections that you can set up include:

- Detect all occurrences in history when the value of a discrete tag is set to 0
- Detect if the system clock is set to a specified date and/or time
- Determine the state of information in the database by a SQL statement

You can use event actions to perform tasks such as the following:

- Send e-mail messages to remind managers about weekly maintenance checks
- Summarize plant data to create a statistical analysis over defined periods of time
- Take "snapshots" of system data
- Modify storage conditions (such as time and value deadbands)
- Generally perform any database-related task

The event subystem is not designed to transfer data to and from the database continually and should not be used in this manner. The only exception is for summary actions; the event subsystem can continually process data aggregates so that they are available for reporting purposes.

The event subsystem should not be used as an alarm system. An alarm system such as provided with InTouch HMI software can be used to alert operators to specific satisfied conditions. The InTouch alarm system is intended as a notification system to inform operators of process and system conditions promptly upon their occurrence. The InTouch alarm system supports displaying, logging, and printing capabilities for process alarms and system events. (Alarms represent warnings of process conditions, while events represent normal system status messages.) For more information on the InTouch alarm system, see your InTouch documentation.

In contrast, the event subsystem is intended to initiate actions based upon historical event detection. An alarm system presupposes an immediate message response is propagated for all configured alarms at the time the respective conditions are met. In this sense, the historian event subsystem is not an alarm system. The event subsystem queues up detected events and processes them accordingly based upon pre-configured priorities.

Event Subsystem Features and Benefits

You can obtain a number of distinct operational benefits from properly using the features of the event subsystem. A list of key benefits is as follows:

- Unlike real-time alarming, the event subsystem determines events from stored historical data and is not dependent on real-time detection. No events are missed unless the machine is severely overloaded for a long period of time.
- The event subsystem is SQL-based, thus providing a means of managing database-related tasks within the system. You can use custom SQL queries as detectors, as well as create custom SQL-based actions.
- A number of pre-configured detectors and actions are available.
- Detections may be made by external sources. (A COM mechanism is available for invoking the detector in the event subsystem.)
- Time-based detection (based on the system clock time) allows you to schedule certain tasks, such as data aggregations (summaries).

- The event subsystem is designed to manage overload situations. If the system is currently busy due to some other processing for a period of time, the event subsystem will "catch up" at a later time during off-peak periods. If the overall Wonderware Historian is continuously overloaded, the event subsystem degrades in functionality gracefully.
- You can select which actions have priority and can assign certain actions (preferably only a few) never to be compromised, even under overload conditions.
- System tags are available to monitor event subsystem conditions.

Event Subsystem Performance Factors

The overall performance of the Wonderware Historian event subsystem is subject to factors related to data storage and query processing time. Too often, systems are commissioned with specifications that estimate average or "typical" expected loading. Instead, you should size the system so that it can accommodate the peak load that you expect during the projected system life cycle. Some performance factors you should consider are:

- Sufficient hardware. Your selection of hardware is important to guarantee peak performance for the range of behaviors required for a given operating environment. For example, you should make sure that you have enough disk space to store the records of detected events and the results of any actions (summaries, value snapshots, and so on).
- Processor availability. The event subsystem is subject to processor availability as much as any other software sharing a common platform. At any given moment, multiple processes contend for processor time.
- Nature of the database queries executed by the event subsystem. For example, because event subsystem actions typically operate on normal SQL Server tables, they are subject to performance limitations of the Microsoft SQL Server. Also, query activity tends to be very CPU-intensive and is extremely sensitive to other concurrent activities being performed on the same server.
- Time intervals for SQL-based detectors. For more information, see "Time Intervals for SQL-Based Detectors" on page 346.

Performance can vary greatly for the same event task, depending upon the computer configuration, user interaction, and other unpredictable activity common in a plant situation with shared database and server resources. It is often very difficult to determine precisely what combinations of hardware and software parameters will work optimally for your required operating environment. Therefore, you should test your event subsystem configuration before running it in a production environment to make sure that the system will not become overloaded during peak use.

Event Tags

An event tag is a name for an event definition in the system. For example, if you want to detect an event when a tank temperature reaches 100 degrees, you can define an event tag and name it "TankAt100." Event tags differ from the other tag types in the Wonderware Historian (analog, discrete, and string). Analog, discrete, and string tag types are the definitions of variables to be stored. In contrast, an event tag is a named reference for the definition of the specific event you want to detect, including an optional action to perform when the event is detected. An event tag provides a way to reference all event definition information in the system.

Event tags are created and maintained using the System Management Console. When you define an event tag, you must specify:

- A name, description, and other general configuration information.
- The event criteria, which describes the conditions that must exist for the event and how often the event subsystem checks to see if an event occurred.
- Whether or not to log the event detection.
- Whether or not to enable or disable event detection.
- An optional action that is triggered when an event is detected.

Event Detectors

Each event tag must have an associated event detector. An event detector is a mechanism for determining when the set of event criteria for an event tag has been satisfied. When you configure an event detector, you must first configure its type and then configure the parameters associated with that detector type. You can choose from the following types of event detectors:

- SQL-Based Detectors
- Schedule Detectors
- External Detectors

The generic SQL, analog specific value, and discrete specific value detectors are SQL-based detectors. The schedule detector is a time-based detector. The external detector is used when triggering an event by the ActiveEvent ActiveX control.

For all detectors, the event subsystem will initially base the query for data in history at the time the event subsystem starts. Subsequently, the event subsystem will base the query on the last successful detection; that is, the time of the most recent detection becomes the starting time for the next detection.

SQL-Based Detectors

Analog specific value, discrete specific value, and generic SQL detectors operate on data stored in the database. The detection criteria for each of these detectors is a SQL statement that is executed against the Wonderware Historian. Generic SQL detectors can query against both the historian and Microsoft SQL Server.

Generic SQL Detectors

A generic SQL detector detects an event based on criteria that are specified in a SQL statement. You can use pre-configured SQL templates that are stored in the database as the basis for your script, or you can create your own script from scratch.

To use a pre-configured SQL template, simply select it from a list of available templates when defining the event tag.

If you create a new script, you will need to add it to the SQLTemplates table in the Runtime database in order for it to appear in the list of pre-configured templates. You should test your SQL queries in SQL Server Query Analyzer before using them in a generic SQL event detector.

Specific Value Detectors

Two specific value detectors are available:

- Analog specific value detector
- Discrete specific value detector

These detectors can be used to detect if a historical tag value matches the state defined by the detector criteria. For the criteria, historical values are compared to a target value that you specify. If a value matches the criteria, then an event is logged into the EventHistory table, and any associated actions will be triggered. For example, an analog specific value detector could be configured to detect if the value of 'MyAnalogTag' was ever greater than 1500. Likewise, a discrete value detector could be configured to detect if the value of 'MyDiscreteTag' was ever equal to 0.

For a specific value detectors, you can apply either edge detection or a resolution to the returned data. The resolution is used only when the edge detection is set to NONE (in which case the retrieval mode is cyclic). For more information, see "Resolution (Values Spaced Every X ms) (wwResolution)" on page 222 and "Edge Detection for Events (wwEdgeDetection)" on page 264.

Time Intervals for SQL-Based Detectors

For SQL-based detectors, you must specify a time interval that indicates how often the detector will execute. The time interval is very important in that it affects both the response rate of any event actions and the overall performance of the system.

The detection of an event may occur significantly later than the actual time that the event occurred, depending on the value you specify for the time interval. The time between when an event actually occurred in history and when it was detected is called latency.

For example, you configure a detector to detect a particular event based on a time interval of 10,000 ms (10 seconds). This means that every 10 seconds, the event detector will check to see if the event occurred. If the event occurs 2,000 ms (2 sec) after the last check, the event detector will not detect that the event occurred until the full 10 seconds has elapsed. Thus, if you want a greater possibility of detecting an event sooner, you should set the time interval to a lower value.

Also, the time interval affects when an associated action will occur, because there could be some actions that are queued to a time equal to or greater than the interval.

The following are recommendations for assigning time intervals:

- When configuring multiple event detectors, distribute them evenly across multiple time intervals; don't assign them all to the same interval.

  All configured detectors are first divided into groups, based on their assigned time interval. The detectors are then sequentially ordered for processing in the time interval group. The more detectors assigned to a particular time interval, the longer it will take the system to finally process the last one in the group. While this should not have a negative impact on actual detection of events, it may add to increased latency.
- Avoid assigning a faster time interval than is really necessary.

  The time interval for detectors should not be confused with a rate required by a real-time system that needs to sample and catch the changes. For the event subsystem, a slower time interval simply means that more rows are returned for each scan of the history data; no events are lost unless then detection window is exceeded (for more information, see "Detector overloads" on page 357). For example, you create an event tag with a detector time interval of 1 minute, and you expect an event to occur every 5 seconds. This means that the system would detect 12 events at each time interval. In most cases, this is an acceptable rate of detection. Also, assigning short time intervals will result in higher CPU loading and may lead to degraded performance.

For detailed information on how detectors are executed, see "Event Subsystem Resource Management" on page 353.

The EventHistory table can be used to determine if too many event tags have the same time interval. If the latency between when the event actually occurs (stored in the DateTime column) and when it was detected (stored in the DetectDateTime column) is constantly growing and/or multiple event occurrences are being detected during the same detector time interval, you need to move some of the event detectors to a different time interval.

Schedule Detectors

The schedule detector is a time-based detector. A schedule detector detects whether the system clock is equal to or greater than a specific date and/or time. For example, you could log an event every week on Monday at 2:00 p.m.

Schedule detectors are different from other detectors in that they are real-time detectors. The value of the system clock is checked every second. Schedule detectors are very fast and can be used without great concern about efficiency. Thus, a schedule detector provides the only real-time event processing. However, there is no guarantee of when the action will occur.

All of the schedule detectors that you set up are handled by a dedicated scheduling thread. This allows for a separation between the processing load needed to execute schedule detectors and the processing load needed to perform all of the other event work. The scheduling thread will maintain a list of detection times in a time queue. If you add a schedule detector, the thread will register the detection time in the queue and then re-sort the list of all detection times from the earliest to the latest.

The time of the system clock is then compared with the time of the first item in the schedule queue. If the system clock time is equal to or greater than the time of the first item, the detection algorithm for the first item will be invoked and the detection will be performed.

The event subsystem does not account for Daylight Savings Time changes. If you set up a schedule detector that runs periodically with a specified start time, you will need to change the start time to reflect the time change. Another solution would be to use the time-weighted average retrieval mode instead of the event subsystem to generate averages, because the retrieval mode handles the Daylight Savings Time changes. However, if the period for the average is hourly, then it is recommended that you use the event subsystem, as the amount of data will not generally not be a factor in the speed of calculating the average.

External Detectors

For an external detector, event detection is triggered from an external source by the ActiveEvent ActiveX control that is provided as part of the Wonderware Historian. For example, an InTouch or Visual Basic script can invoke the necessary ActiveEvent methods to trigger an event. This ActiveX control must be installed on the computer from which you want to trigger the external event.

For more information, see "Configuring an External Detector" in Chapter 11, "Configuring Events," in your *Wonderware Historian Database Reference*.

Event Actions

An event may or may not be associated with an event action. An event action is triggered after the event detector determines that the event has occurred. The event subsystem is not intended to run external processes. There is only a very limited ability to run external program files or to call methods from COM interfaces within the given system or network.

Actions are not required; there are times when you may want to simply store when events happened. In this case, you would select "None" for the action type when defining the event tag.

Generic SQL Actions

A generic SQL action executes an action that is outlined in a SQL statement. For example, a SQL action can update the database (for example, turning off storage for tags) or copy data to a separate table or database.

You can use pre-configured SQL templates that are stored in the database as the basis for your script, or you can create your own script entirely from scratch. You cannot submit multiple queries against the Wonderware Historian in a single event action and you cannot use GO statements. Also, if you are querying against history data, the SQL statement is subject to the syntax supported by the Wonderware Historian OLE DB provider. You should test your SQL queries in SQL Server Query Analyzer before using them in a generic SQL event action.

Snapshot Actions

A snapshot action logs into dedicated SQL Server tables the data values for selected analog, discrete, or string tags that have the same timestamp as the detected event. Quality is also logged. Value snapshots are stored in tables according to the tag type, either AnalogSnapshot, DiscreteSnapshot, or StringSnapshot.

A snapshot action requires an expensive SQL join between the extension tables and the snapshot tag table. The process of performing the join and logging the retrieved results to the snapshot tables can be very slow. This is because most of the tables used for event snapshots are normal SQL Server tables, subject to the data processing limitations of Microsoft SQL Server. Thus, the higher the number of snapshots that are being taken by the event system, the higher the transaction load on the Microsoft SQL Server.

Important The event subsystem is not a data acquisition system. DO NOT attempt to use snapshot actions to move data stored in the extension tables to normal SQL Server tables. This type of misapplication is guaranteed to result in exceedingly poor throughput and storage rates.

When trying to determine how many snapshots can be made by the system, you should execute the intended snapshot queries to the server using a batch file, leaving the event subsystem out of the exercise. By executing repeated snapshot queries at the server as fast as the computer will allow, you can better determine how many snapshots can be performed on a system over a given time period. Using this result and applying a safety factor may provide a good guideline for assessing how much your system can safely handle. Keep in mind that discrete snapshots are many times slower than analog snapshots.

E-mail Actions

An e-mail action sends a pre-configured Microsoft Exchange e-mail message. Although e-mail actions are useful for sending non-critical messages triggered by an event detection, these types of actions are not to be used for alarm-type functionality. For e-mail notifications of alarm situations, use an alarm system such as the SCADAlarm alarm notification software.

Deadband Actions

A deadband action changes the time and/or value storage deadband for one or more tags that are using delta storage. (Value deadbands only apply to analog tags.) Deadband change actions are useful for increasing data storage based on an event occurring. For example, an event detector has detected that a boiler has tripped, you might want to start saving the values of certain tags at a higher rate to help you determine the cause of the trip.

Summary Actions

A summary action is a set of aggregation calculations to be performed on a set of tags between a start time and an end time with a defined resolution. When you configure a summary action, you must define the type of aggregation you want to perform (called a summary operation) and the analog tags that you want to be summarized. The event subsystem performs average, minimum, maximum and sum calculations on the basis of a specific event being detected.

Note Summary actions using the event subsystem are retained for backward compatibility. We recommend that you use the more robust and flexible replication subsystem to perform data summaries. For more information, see Chapter 9, "Replication Subsystem."

Data summaries are useful for:

- Extremely long-term data storage. Because summarized data takes up less space than full resolution data, even a moderately sized system can store daily summary information for many years.
- Production reporting. For many reporting purposes, aggregate data is more important than raw data. For example, the total mass produced in a day is often more relevant than the actual rate of production during the day.
- Integration with business systems. The full resolution, high-performance Wonderware Historian history and real-time data tables are best accessed with tools that can take advantage of the Wonderware Historian time domain extensions. However, not all client tools support these SQL extensions. The summary tables reduce the volumes of data to manageable quantities that can be used by any normal SQL client application.

A summary action is usually triggered by a schedule detector. However, you can perform a summary as a result of any event detection.

Tag values with bad quality are not filtered out before the aggregation is performed. To perform an aggregation with only good quality, for example, use a generic SQL action that executes an aggregation calculation query on the History table where the value of the Quality column equals 0.

The results of all summaries are stored in the SummaryData table in the Runtime database.

Important Use caution when setting up summary actions. Using a high resolution for your summary queries can have a negative impact on the overall performance of the system.

Average, minumum, and maximum values can also be determined by using the time-weighted average, minimum, and maximum retrieval modes, respectively. For more information on these retrieval modes, see "Understanding Retrieval Modes" on page 151. Keep the following in mind when deciding to use either the event summaries or the retrieval modes:

- For the time-weighted average retrieval mode, the amount of time between the data values is a factor in the average, whereas the event summary action is a straight statistical average. For more information, see "Average Retrieval" on page 176.
- Performing an average at retrieval eliminates problems that occur during Daylight Savings Time adjustments for schedule-based summaries. For more information, see "Schedule Detectors" on page 348.

For a comparison of all the different types of summaries that the Wonderware Historian supports, see Querying Aggregate Data in Different Ways on page 316.

Event Action Priorities

The event subsystem contains three different queues for event actions:

- A "critical" queue, which contains any actions for event tags that have been assigned a critical priority. Actions for events that are given a critical priority will be processed first. It is extremely important that the critical queue is used with caution. Only singularly important actions with short processing times should be assigned as critical. You should never assign snapshot or summary actions as critical. There is no overload protection for processing critical actions; if the system becomes overloaded, actions may not execute in a timely fashion or may not execute at all.
- A "normal" queue, which contains any actions for event tags that have been assigned a normal priority. All non-critical events are labeled with a "normal" priority and will be processed after the critical events.
- A “delayed action” queue, which contains any actions for event tags that have been assigned a post-detector delay. The post detector delay is the minimum amount of time that must elapse after an event was detected before the associated action can be executed.

Event Subsystem Resource Management

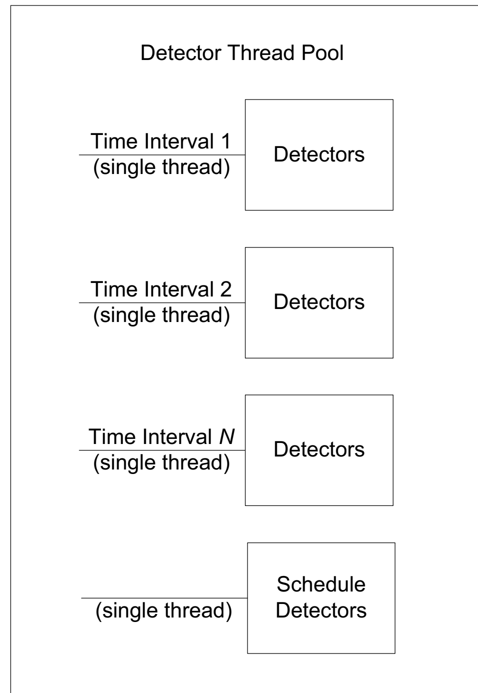
The Event System Service (aahEventSvc.exe) manages all of the system resources required to detect events and process actions. System resources are allocated for detectors and actions by means of threads. A thread is an operating system component that independently performs a particular function within a larger process. Within the overall process of the event subsystem, event detectors and actions are assigned different threads, so that they can execute independently of each other and thus perform more efficiently.

The event subsystem uses two thread groups, or "pools." One thread pool is for detectors and the other one is for actions. The Event Service automatically creates both of these thread pools if there is at least one event tag defined.

Other aspects of resource management include the number of database connections required by event system components, and how the system handles event overloads and query failures.

Detector Thread Pooling

The detector thread pool is made up of one or more threads allocated for SQL-based detectors and a single thread for schedule detectors. Each thread maintains a connection to the database. The detector thread pool is illustrated in the following diagram:



A SQL-based detector is assigned to a thread based on the time interval that you specify when you define the event tag. Each time interval requires its own thread. For example, you define three event detectors and assign them time intervals of 10, 15, and 20 seconds, respectively. Each event detector will be running in its own thread, for a total of three threads.

As another example, you define three event detectors, assigning the first two a 10 second interval, and the third a 15 second interval. The first two will be running under the same thread, while the third will be running under its own thread, for a total of two threads.

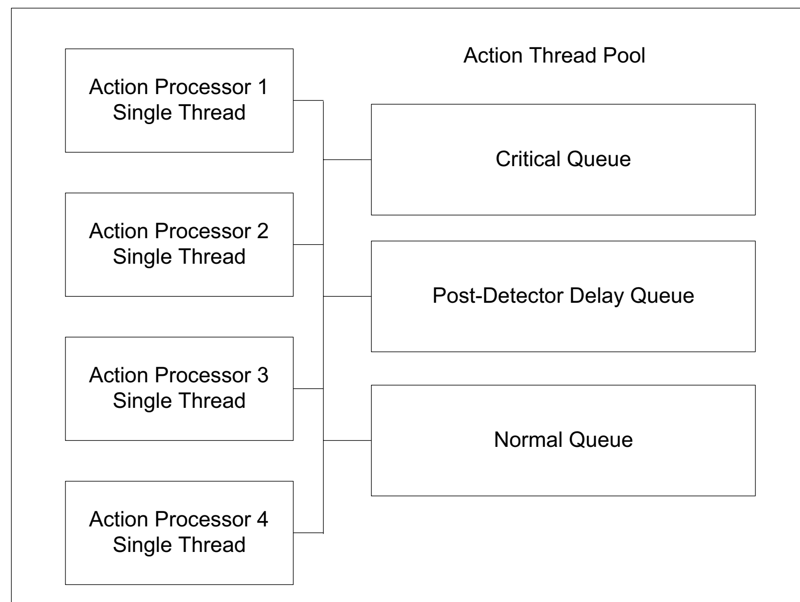
For multiple detectors that are assigned to the same time interval, the SQL detection statement for each event tag will be executed in sequential order. That is, the first SQL statement must return results before the next statement can be executed. After each detection has taken place (results are returned), the detection is logged into the EventHistory table and any associated action is queued into the action thread pool.

All schedule detectors are assigned to a single thread.

The efficiency of the detector thread pool depends on how you have spread the load when assigning time intervals to different event tags. Detections generally do not cause overloading on the system: the actions (especially snapshots and summaries) are where most processing and resource loading occurs.

Action Thread Pooling

The action thread pool is essentially a pool of four threads that execute actions from three different action queues. Each thread in the pool maintains a database connection.



The three action queues are:

- Critical queue
- Normal queue
- Post-detector delay queue

For detailed information about each of these queues, see "Event Action Priorities" on page 353.

As a processor thread completes its previous task, a new action will be fetched from one of the queues. If there are any actions in the critical queue, these will be processed first. Actions in the critical queue are executed in the order in which they were added to the queue; that is, the oldest action sitting in the queue will be processed first.

If the critical queue is empty, actions will be fetched from the post-detector delay queue. Actions in the post-detector delay queue are ordered by time. Actions assigned the shortest post-detector delay will be executed first.

If both the critical and post-detector delay queues are empty, actions will be fetched from the normal queue. Like critical actions, normal actions are processed in the order in which they were added to the queue.

Event Subsystem Database Connections

The following table contains the number of SQL Server database connections required by the different components of the event subsystem.

Component	Number of Connections Used
Event Service	1
SQL-based detectors	1 per each time interval used
Schedule detectors	1
Action threads	4

Handling of Event Overloads and Failed Queries

The event subsystem handles SQL-based detector and action queries that fail, as well as to degrade gracefully if detector and action overload conditions occur.

- Event query failures

  If the query for a SQL-based detector fails, the query will automatically be executed again. The detection window start time will remain the same until the next detection is made.

  For a failed SQL-based action query, the query will be submitted three times. The system will establish a new connection to the database each time the query executes. If the action query is a snapshot query, the snapshot tables will first be "cleaned up" as part of the re-query process.

- Detector overloads

  A detector overload occurs when the system cannot process all of the detectors in a timely manner. Detector overload is handled by means of the detection window. This window is defined by the difference between the current system time and the time of the last detection. If the window grows larger than one hour, some detections will be missed. This condition will be reported in the error log.

- Action overloads

  An action overload occurs when the system cannot process all of the actions in a timely manner. Only actions assigned a normal priority have overload protection. An action will not be loaded into the normal queue by a detector if the earliest action currently sitting in the queue has been there for an hour. (Basically, it is assumed that the system has become so overloaded that it has not had the resources to process a single action in the past hour.) This prevents an accumulation of actions in the normal queue when the system is unable to process them. The system will be allowed time to recover, and actions will not start to be queued again until the time difference between earliest and latest action in the queue is less than 45 minutes (75 percent of the time limit). In short, when the system becomes too overloaded, actions are not queued. This condition is reported in the error log, but not for every single action missed. The first one missed is reported, and thereafter, every hundredth missed action will be logged.

  There is no overload protection for critical actions, because these types of actions should only be configured for a very small number of critical events. There is also no overload protection for actions that have been assigned a post-detector delay.

For more information on action priorities, see "Event Action Priorities" on page 353. For more information on how actions are queued, see "Action Thread Pooling" on page 355.

Event Subsystem Variables

The event subsystem uses a set of variables to facilitate event detections and actions. The purpose of these variables is to provide ease of query creation by a user (or a configuration editor). These variables are replaced with the associated values by the event components immediately before actual query execution. The query actually being received by the Wonderware Historian never contains the variables.

The variables and their associated values are as follows:

Variable	Description/Associated Value
@EventTime	Date/time of the detected event of the current detector.
@EventTagName	Tagname associated with the detected event.
@StartTime	Start date/time for the detector query.
@EndTime	End date/time for the detector query.

The @StartTime and @EndTime variables can be used only in detector strings. The @EventTime and @EventTagName variables can be used only in action strings.

All of the variables are case-sensitive.

Typically, a detection query executed by a detector component is similar to the following example:

```
SELECT DateTime
   FROM History
      WHERE Tagname = 'BoilerPressure' AND Value > 75
      AND DateTime > '@StartTime'
      AND DateTime < '@EndTime'
```

@StartTime and @EndTime are simply placeholders for the detector component to coordinate event detection over a moving time range.

The following action query show how event variables can be used:

```
SELECT * INTO TEMPTABLE
    FROM History
        WHERE DateTime = '@EventTime'
        AND TagName IN
        (SELECT TagName FROM SnapshotTag
        WHERE EventTagName = '@EventTagName'
        AND TagType = 1)
```

Note These variables only function in the internal context of the event subsystem and do not apply to queries from client tools such as SQL Server Query Analyzer.

Index

Symbols

< (less than) 294, 298, 300, 301, 305
<= (less than or equal to) 294, 298, 299, 300, 302
> (greater than) 294, 296, 300, 301, 303, 305
>= (greater than or equal to) 294, 299, 300, 302, 303

Numerics

16-bit 125
32-bit 112, 125
64-bit 112, 125

A

aaAdmin 27
aadbo 27
aaPower 27
aaUser 27
acquisition
  about 17
  components 72
  CSV data 91
  I/O Servers 73
  IDASs 75
  MDAS clients 90
  quality 59
  service 40
  sources of data 71
  Transact-SQL 90
actions
  See also event actions
active image 120
  about 119
  automatic resizing 120
  resizing 33
  retrieval 315
  storage 94, 121, 334
ActiveEvent 349
ActiveX control 349
aggregate functions 292
aggregations 351
AIAutoResize system parameter 33
AIResizeInterval system parameter 33
alarm system 342
AllowOriginals system parameter 33
alternate storage location 115, 116
analog specific value detector 345, 346
analog summary replication
  about 325
analog tags
  about 21
  edge detection 265, 266, 267, 268
AnalogSummaryHistory table 132
AnalogSummaryTypeAbbreviation system parameter 33

annotations
  making 294
application name 74
ArchestrA
  user account 24
architecture 19
  Wonderware Historian 19
arithmetic functions 291
authentication
  databases 25
  SQL Server 25, 26
AutoStart system parameter 34
average (time-weighted), retrieval modes 176
averages 176

B

best fit retrieval mode 171
best fit, retrieval modes 171
blocks
  See history blocks
buffer storage location 115, 117

C

cache
  IDAS 79
catalog 134
circular storage location 115
  about 116
client
  quality 61
client applications 15
  support 19
client context 138
client/server 19
comparison operators 294, 299, 302
ConfigEditorVersion parameter 34
configuration
  modification tracking 52
  service 40
  See also dynamic configuration
configuration data 16, 63
Configuration Service 64, 334
  about 67
configuration subsystem
  components 64
configuration tables 63
CONVERT function 142, 287
COUNT(*) function 290
counter retrieval mode 200
counter, retrieval modes 200
criteria condition 287
CSV data
  acquisition 91
  import path 34
  manual data 22
  quality 60
cursors 308
cycle count 219, 299
cyclic retrieval
  using comparison operators 299, 302
cyclic retrieval, about 152
cyclic storage 99
  about 111
cyclic, retrieval modes 152

D

data acquisition
  See acquisition
data blocks
  See history blocks
data files
  See history blocks, Runtime database
data quality
  See also quality
data retrieval
  See retrieval
data sources 129
data storage
  See storage
data stores 129
database
  authorization 25
database authorization 28
database connections
  for events 356
DatabaseVersion system parameter 34
DataImportPath system parameter 34
date functions 288
Daylight Savings Time 328
DCOM 90
DDE 50, 73
  about 50
deadband 100, 227, 231, 282
  swinging door 101
  time 100

value 100
deadband actions 351
deadband override period 104
delayed action queue 353
delta retrieval
comparison operators 294
querying wide tables 278
delta retrieval, about 156
delta storage 99
about 100
delta, retrieval modes 156
detectors
See event detectors
discrete specific value detector 345, 346
discrete tags
about 21
edge detection 268, 270, 271, 272
disk space
plant data 18
document conventions 12
documentation set 11
dynamic configuration
about 67
committing changes 70
effects 68
storage 122

E

edge detection 264
analog tags 265, 266, 267, 268
discrete tags 268, 270, 271, 272
e-mail actions 350
EndTime variable 358
engineering units 231
error count
system tags 42
error messages 38
categories 39
See also system messages
errors
IDAS 78
event actions
about 349
deadband actions
about 351
e-mail actions
about 350
generic SQL actions
about 349
overloads 357
priorities 353
queues 355
snapshot actions
about 350
summary actions
about 351
thread pooling 355
event data 16
event detectors
about 345
external detectors
about 349
generic SQL detectors
about 345
overloads 357
schedule detectors
about 348
specific value detectors
about 346
SQL-based detectors
about 345
thread pooling 354
event history
storage duration 34
event subsystem
about 339
Event System Service 40, 340
about 353
event tags
about 344
events
about 339
components 340
database connections 356
edge detection 264
features and benefits 342
overloading 356
performance factors 343
uses 341
variables 358
See also event actions, event detectors
EventStorageDuration system parameter 34
EventTagName variable 358
EventTime variable 358

extension tables 147, 218
  about 132
external detectors 349

F

failover
  I/O Servers 86
  IDAS 80
FastDDE 73
FILETIME 129, 310
firewall
  IDAS 78
forced storage 99
  about 99
four-part naming convention 133
full, retrieval modes 163

G

generic SQL actions
  about 349
generic SQL detectors 345
  about 345
Greenwich Mean Time 32
GROUP BY clause 137, 290
groups
  security 26

H

handle 49
Headroom system parameter 34
heterogeneous query 131
Historian Configuration service 40
Historian Console
  See also Management Console
Historian DataAcquistion service 40
Historian EventSystem service 40
Historian I/O Server
  about 149
  service 40
Historian IOServer service 40
Historian ManualStorage service 40
Historian MDASServer service 41
Historian OLE DB Provider
  See OLE DB provider
Historian Replication service 41
Historian Retrieval service 41
Historian SCM service 41
Historian Storage service 41
Historian SystemDriver service 41
HistorianVersion parameter 34
historical data 15, 16
history blocks
  about 113
  component of storage 94, 334
  creating 114
  deleting 117
  duration 35
  notation 113
  querying data 313
  remote data source 129
  retrieval 315
  storage locations 115
history data
  acquisition 72
  data categories 94
  modification tracking 53
History table 132
  inserting manual data 307
  querying 275
history version 235
HistoryCacheSize parameter 35
HistoryDaysAlwaysCached parameter 35
Holding database
  about 66
HoursPerBlock system parameter 35

I

I/O Driver 75
I/O Servers
  acquisition 17, 72, 73
  addressing 74
  data quality 59
  failover 86
  inserting original data 33
  modification effects 70
  redirecting to InTouch 87
  time synchronization 37, 87
IDASs
  about 75
  acquisition 72, 75
  configuration information 76
  data processing 77
  data transmission 77
  error logging 78
  failover 80

late data handling 82
licensing 35
modification effects 70
performance 48
security 78
slow and intermittent networks 85
store-and-forward 79
system tags 44
time synchronization 37, 38, 87
importing
Holding database 66
IN clause 138
indexing 40
indexing service 122
INNER REMOTE JOIN 140, 281
INSERT statements 90
InSQLIOS
See Historian I/O Server
integral retrieval mode 194
integral, retrieval modes 194
intermittent network. 85
interpolated, retrieval modes 165
interpolation 35, 165
interpolation type 237
InterpolationTypeInteger parameter 35
InterpolationTypeReal parameter 35
InTouch
redirecting I/O Server data 87
InTouch History Importer 72
item name 74

J

JOIN clause 139, 140, 281

L

late data 82
characteristics 94
LateDataPathThreshold system parameter 35
latency 336
event detections 346
leading edge detection 266, 268, 270, 272
LicenseRemoteIDASCount parameter 35
LicenseTagCount parameter 35
licensing
remote IDASs 35
tag count 35
LIKE clause 138
linear interpolation 238
linear scaling 130
linked server 133, 146
Live table 132
querying 276
log file
See system message logs
logins
Windows 24
Wonderware Historian 27
Wonderware Historian services 31

M

Management Console
security 31
ManualDataPathThreshold system parameter 35
maximum retrieval mode 188
maximum, retrieval modes 188
MDAS
acquisition 72, 73
clients
acquisition 90
retrieval 130
time synchronization 89
data quality 60
retrieval 128
storage 94, 334
MDAS Server Service 334
memory
history information 94
management 122
tag information 35
Microsoft Query 145
Microsoft SQL Server
integration 18
security 25
minimum, retrieval modes 182
mixed mode 26
modification tracking
about 51
configuration changes 52
historical data changes 53
turning on or off 36
ModLogTrackingStatus system parameter 36

monitoring
  performance tags 38
  system tags 43, 44

N

net time command 89
network
  IDAS 85

O

Object Linking and Embedding for Databases (OLE) 130
object permissions 28
ODBC 19
old data
  characteristics 94
  replication delay 336
OldDataSynchronizationDelay system parameter 36
OLE DB 130
OLE DB provider
  about 130
  four-part query 133
  limitations 137
  linking 146
  retrieval 128
  stored procedures 310
  syntax supported 137
OPC quality 244, 312
  about 54
OPENQUERY function 135
OPENQUERY statement 139, 145, 312
OPENROWSET function 136
operations
  See summary operations
operators 294, 299, 302
OR clause 139
ORDER BY clause 137
original data 98, 235
overloading 356, 357

P

page faults 49
performance
  system tags 38, 48
Performance Logs and Alerts 48
permanent storage location 115, 117
permissions
  security 28
post detector delay 353
priority
  event actions 353
process data 16
production data 15
protocols 50, 73

Q

quality
  about 54, 61
  client-side 61
  data acquisition 59
  retrieval rule 244
  viewing values for 55
quality detail codes 56
quality flags 58
quality rule 244
  parameter 36
QualityDetail
  about 54
QualityDetail bit layout 59
QualityRule parameter 36
query
  examples 275
  OLE DB provider syntax 133, 137
queued replication 332
queues
  event actions 353

R

rate of change 101
real-time
  data storage 94, 334
real-time data 16
  characteristics 94
real-time data window 95
  about 97
  late data 83
  parameter 36
  swinging door 104, 110
RealTimeWindow parameter 36
reconfiguration
  See dynamic configuration
redirect I/O Servers 87
redundancy
  See failover

relational databases 15, 65
  about 16
  limitations 17
  Wonderware Historian 17
remote data source 129
remote table
  See also extension tables
replication
  about 319
  analog summary 325
  comparison to event system 338
  continuous operation 336
  data 331
  Daylight Savings Time 328
  delay for old data 336
  latency 336
  overflow protection 337
  queued 332
  run-time operations 335
  security 337
  state summary 326
  streaming 332
  tags 321
replication groups
  about 330
replication schedules
  about 327
Replication Service 334
replication subsystem 334
ReplicationConcurrentOperations system parameter 36
ReplicationDefaultPrefix system parameter 36
ReplicationTCPPort system parameter 37
resolution 15, 222, 310
  using comparison operators 302
retrieval
  about 127
  active image 121
  components 128
  deadbands 282
  edge detection 264
  features 129
  quality rule 244
  service 41, 130
  time deadband 227
  time zone 242
  value deadband 231
  version of data 235
  See also cyclic retrieval, delta retrieval
retrieval modes
  average (time-weighted) 176
  best fit 171
  counter 200
  cyclic 152
  delta 156
  full 163
  integral 194
  interpolated 165
  maximum 188
  minimum 182
  slope 197
  time-in-state 205
  ValueState 205
retrieval service 128
RevisionLogPath system parameter 37
roles
  permissions 29
  Wonderware Historian 29
rollover value 200
RTU 83
Runtime database
  about 65
  configuration data 63

S

samples 119
sampling interval 222
schedule detectors
  about 348
schema 134
security 19
  about 24
  database authorization 28
  IDAS 78
  Management Console 31
  permissions 28
  replication 337
  roles
    Wonderware Historian defaults 29
  SQL Server authentication 26
  SQL Server security 25
  users
    Wonderware Historian default 29

Windows groups 26
Windows operating system 24
Wonderware Historian logins 27
Wonderware Historian services 24
SELECT INTO statement 306
SELECT statements 140
syntax 133
services
security for 24
SQL Server login 31
Wonderware Historian 40
simple replication
about 323
SimpleReplicationNamingScheme system parameter 37
slope retrieval mode 197
slope, retrieval modes 197
slow network 85
snapshot actions 350
snapshot files
about 124
updates 126
specific value detectors 346
SQL 16
SQL Server
authentication 26
See Microsoft SQL Server
SQL Server logins
See logins
SQL statements
for actions 349
SQL templates 345
SQL-based detectors 345
time intervals 346
stairstep interpolation 237
StartTime variable 358
state summary replication
about 326
statement permissions 29
StateSummaryHistory table 132
StateSummaryTypeAbbreviation system parameter 37
statistical average 176
storage
about 93
active image 121
components 94
data categories 94
data conversions 112
data versioning 98
dynamic configuration 122
forced storage 99
methods 99
real-time data window 97
reserved values 112
service 40
system tags 43
Wonderware Historian 18
See also delta storage, cyclic storage
See also history blocks, storage locations, delta storage, cyclic storage
storage by exception
See also delta storage
storage deadband 351
storage locations
about 115
alternate 116
buffer 117
circular 116
modification effects 69
path 115
permanent 117
storage rate
cyclic storage 111
store-and-forward
as manual data 22
IDASs 79
stored procedures
OLE DB queries 310
string tags
about 21
SuiteLink 17, 73
about 50
SuiteLinkTimeSyncInterval system parameter 37
summary actions
about 351
summary data 16
summary history
storage duration 38
summary operations 351
summary replication
about 324
summary tags
about 22

SummaryCalculationTimeout system parameter 37
SummaryReplicationNamingScheme system parameter 38
SummaryStorageDuration system parameter 38
swinging door deadband 101
  benefits 101
  examples 105
  options 104
  real-time data window 110
  requirements 110
SysPerfTags system parameter 38
system driver
  about 42
  acquisition 73
  service 41
System Management Console 64, 334
system messages
  about 38
  categories 39
system parameters
  about 33
  modification effects 69
system tags
  about 42

T

tags
  about 21
  allocating memory 34
  conventions 281
  information in memory 122
  mixing in a query 286
  modification effects 69
  replication configuration 333
  sources of values 22
  types 21
threads 49
  event actions 355
  event detector 354
  event system 353
tiered historian
  about 319
time deadband 100, 227, 282
  event action 351
time deadband, retrieval 227
time domain extensions 18, 147, 218
time handling 32
  query resolution 310
  synchronization 37
time interval
  SQL-based detectors 346
time series data 18
time synchronization
  I/O Servers 87
  IDASs 87
  MDAS clients 89
  Wonderware Historian 38, 87
time zone 32, 242
time-in-state 252
time-in-state retrieval mode 205
time-in-state, retrieval modes 205
timestamp rule 240
  parameter 38
timestamping 315
TimeStampRule parameter 38
TimeSyncIODrivers system parameter 38
TimeSyncMaster system parameter 38
time-weighted average retrieval mode 176, 194
topics
  I/O Servers 74
  modification effects 70
tracking
  See modification tracking
trailing edge detection 267, 268, 271, 272
Transact-SQL 18, 90, 147, 218

U

UPDATE statements 90
user account
  See logins
users
  Wonderware Historian 29
UTC 32, 242

V

value deadband 100, 231, 282
  event action 351
  option for swinging door 104
value deadband, retrieval 231
ValueState retrieval mode 205
ValueState, retrieval modes 205

variables 312
  events 358
variant type data 287
version
  stored data 98
version of data, retrieval 235
versioned data 235
views
  using 135

W

WHERE clause 141
wide tables
  delta retrieval 278
  limitations 138
  querying 281
  using variables 312
WideHistory table 132
  querying 277
Windows authentication 26
Windows login
  See logins
Windows operating system
  security 24
Windows security groups 26
Windows Server 2003 50, 73
Windows Server 2008 50
Windows Vista 50
Wonderware Historian
  architecture 19
  documentation set 11
  integration with Microsoft SQL Server 18
  logins 27, 31
  roles 29
  security 24, 25
  services 40
  starting 34
  storage 18
  subsystems 19
  time synchronization 38, 87
  users 29
  version 34
Wonderware Indexing service 40
wwAdmin login 28, 30
wwAdministrators role 30
wwCycleCount column 147, 285
wwdbo user 28, 30
wwEdgeDetection column 147
  about 264
wwFilter column 147
wwInterpolationType column 147
wwParameters column 147, 218
wwPower login 28, 30
wwPowerUsers role 30
wwQualityRule column 147
wwResolution column 147, 285
wwRetrievalMode column 147, 285
wwStateCalc column 147
wwTimeDeadband column 147
wwTimeStampRule column 147
wwTimeZone column 147
  about 242
wwUser login 28, 30
wwUsers role 30
wwValueDeadband column 147
wwValueSelector column 147
wwVersion column 147