LISTEN.
THINK.
SOLVE.℠

INTEGRATED PRODUCTION & PERFORMANCE SUITE

Data Management

FactoryTalk® Historian SE

PI SERVER APPLICATION USER'S GUIDE

PUBLICATION HSEPIS-UM020A-EN-E–September 2007

ALLEN-BRADLEY • ROCKWELL SOFTWARE

Rockwell Automation

Version: 9.00.05

PREFACE – USING THIS GUIDE

About this Guide

The ***PI Server Applications User Guide*** explains how to use **PI Server Applications**. PI Server Applications are supplemental subsystems that you can run in conjunction with the PI Server to provide additional functionality. PI Server Applications are not necessary to run the PI Server, and are typically licensed separately.

The **PI Server Applications** included in this guide are:

- ❑ **PI Performance Equations Scheduler**
- ❑ **PI Performance Equations Recalculator**
- ❑ **PI Steam Functions Module**
- ❑ **PI Batch Database**
- ❑ **PI Totalizer Subsystem**
- ❑ **PI Alarm Subsystem**
- ❑ **PI Real-Time Statistical Quality Control (SQC)**

This guide provides full administration and end-user instructions for the PI Server Applications listed above. An additional PI Server Application, the **PI Batch Generator Interface** (**PIBaGen**), is not discussed in this guide. For information regarding **PIBaGen**, see the ***PI Batch Generator User Guide.***

Installation Note

The PI Server and Server Applications are distributed together as one installation. Your ability to use any one of the PI Server Applications depends upon your EULA (End-user License Agreement.) Contact Rockwell Automation Technical Support for licensing information.

The PI Server Documentation Set

The PI Server Documentation Set includes seven user guides, described below. These documents are included on the FactoryTalk Historian SE installation CD under Redist > Docs

Title	Subject Matter
Introduction to PI System Management	A guide to the PI Server for new users and administrators. It explains PI system components, architecture, data flow, utilities and tools. It provides instruction for managing points, archives, backups, interfaces, security and trusts, and performance. It includes a glossary and resource guide.
PI Server Installation and Upgrade Guide	A guide for installing, upgrading and removing PI Servers on Windows and UNIX platforms, including cluster and silent installations.
PI Server System Management Guide	An in-depth administration guide for the PI Server, including starting and stopping systems, managing the Snapshot, Event Queue and Data Archive, monitoring system health, managing backups, interfaces, security, and moving and merging servers. Includes comprehensive instructions for using command-line tools: PIConfig, PIDiag, and PIArtool, and in-depth troubleshooting and repair information.
PI Server Reference Guide	A comprehensive reference guide for the system administrator and advanced management tasks, including: databases; data flow; PI Point classes and attributes, class edit and type edit; exception reporting; compression testing; security; SQL subsystem; PI time format; and overviews of the PI API, and PI-SDK System Management Tool (SMT).
Auditing the PI Server	An administration guide that explains the Audit Database, which provides a secure audit trail of changes to PI System configuration, security settings, and Archive Data. It includes administration procedures to enable auditing, to set subsystem auditing mode, to create and archive database files, and to export audit records.
PI Server Applications User Guide	A guide to key add-on PI Server Applications: Performance Equations (PE), Totalizer, Recalculator, Batch, Alarm, and Real-Time SQC (Statistical Quality Control). Includes a reference guide for Performance Equations, and Steam calculation functions.
PINet and PIonPINet User Guide	A systems administration guide, including installation, upgrade and operations, for PINet for OpenVMS and PIonPINet, which support migration and interoperability between PI2 and PI3 Systems.

Conventions Used in this Guide

This guide uses the following formatting and typographic conventions.

Format	Use	Examples
Title Case	▪ PI Client Tools ▪ PI System Elements ▪ PI Server Subsystems	▪ Use the client tool, FactoryTalk Historian ProcessBook, to verify that all data has been recovered. ▪ All incoming data is queued in the Event Queue by the Snapshot Subsystem.
Italic text	▪ Files, Directories, Paths ▪ Emphasis ▪ New Terms ▪ Fields ▪ References to a chapter or section	▪ The backup script is located in the *\PI\adm* directory. ▪ Archive files can be either *fixed* or *dynamic*. The archive receiving current data is called the *Primary Archive*. ▪ See Section 4.2, *Create a New Primary Archive.*
Bold Italic text	▪ References to a publication	▪ See the ***PI Server Reference Guide***.
Bold text	▪ System and Application components: ▪ Subsystems ▪ Tools / Utilities ▪ Processes / Scripts / Variables ▪ Arguments / Switches / Options ▪ Parameters / Attributes / Values ▪ Properties / Methods / Events / Functions	▪ The Archive Subsystem, **piarchss**, manages data archives. **Piarchss** must be restarted for changes to take effect. ▪ On UNIX, invoke site-specific startup script, **pisitestart.sh**, and on Windows, invoke **pisrvsitestart.bat**. ▪ Three Point Database attributes affect compression: **CompDev**, **CompMin**, and **CompMax**. These are known as the *compression specifications*.
	▪ Procedures and Key Commands	▪ On the **Tools** menu, click **Advanced Options**. ▪ Press **CTRL+ALT+DELETE** to reboot
	▪ Interface components ▪ Menus / Menu Items ▪ Icons / Buttons / Tabs ▪ Dialog box titles and options	▪ Click **Tools** > **Tag Search** to open the **Tag Search** tool. ▪ Click the **Advanced Search** tab. ▪ Use the search parameters **PImean Value = 1**.

`Monospace type: "Consolas" font`	`Consolas monospace` is used for:
	▪ Code examples
	▪ Commands to be typed on the command line (optionally with arguments or switches)
	▪ System input or output such as excerpts from log files and other data displayed in ASCII text
	▪ **`Bold consolas`** is used in the context of a paragraph
		To list current Snapshot information every 5 seconds, use the **`piartool -ss`** command. For example:
		`$ piartool -ss`
		`  Counters for 7-Aug-05 14:35:56`
		`                   Point Count:    10033  0`
		`               Snapshot Events:  1228011  0`
		`  Out of Order Snapshot Events:      130  0`
		`          Snapshot Event Reads:      392  0`
		`          Events Sent to Queue:   771952  0`
		`  Primary Capacity Remaining:    2540349  0`

Light Blue - Underlined	Links to URL / Web sites, and email addresses	http://support.rockwellautomation.com

Related Documentation

Rockwell Automation provides a full range of documentation to help you understand and use the PI Server, PI Server Interfaces, and PI Client Tools. Each Interface has its own manual, and each Client application has its own online help and/or user guide.

The ***UniInt End User Manual*** describes the **OSIsoft Universal Interface (UniInt)**, which is recommended reading for PI Server system managers. Many PI Interfaces are based upon **UniInt**, and this guide provides a deeper understanding of principals of Interface design.

Using PI Server Tools

The PI Server provides two sets of powerful tools that allow system administrators and users to perform system administration tasks and data queries.

- ❑ The **PI Server** includes many command-line tools, such as **pidiag** and **piartool**. The PI Server Documentation Set provides extensive instruction for performing PI Server administrative tasks using command-line tools.
- ❑ The **PI System Management Tools** (**SMT**) is an easy-to-use application that hosts a variety of different plug-ins, which provide all the basic tools you need to manage a PI System. You access this set of tools through a single host application. This host application is sometimes referred to as the SMT Host, but it is more commonly called **System Management Tools** or **SMT**.

In addition to extensive online help that explains how to use all of the features in the **SMT**, the **SMT** includes the ***Introduction to PI System Management*** user guide.

QUICK TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

Chapter 1. PI SERVER APPLICATIONS

1.1 PI Server Applications Overview

The power of the PI System is enhanced by the PI Server Applications, which work on top of the PI Server. PI Server Applications are a set of processing tools that help you get more out of your data by automating specific processes.

The PI Server Applications and reference sections included in this guide are:

Chapter 2 – PI Performance Equations Scheduler

Chapter 3 – PI Performance Equations Recalculator

Chapter 4 – PI PE Syntax and Functions Reference

Chapter 5 – PI Steam Functions Reference

Chapter 6 – PI Batch Database

Chapter 7 – PI Totalizer Subsystem

Chapter 8 – PI Alarm Subsystem

Chapter 9 – PI Real-Time Statistical Quality Control (SQC)

1.1.1 PI Performance Equations Scheduler

The **Performance Equations (PE) Scheduler** provides an equation syntax and library of built-in functions that allow you to perform a wide variety of calculations on PI data.

Performance Equations can work with frequently-updating Snapshot and Archive values, whereas tools such as spreadsheets only have access to Archive data and limited access to Snapshot update values.

Each Performance Equation is associated with a PI point, and the calculation results are stored in the PI Archive. The performance equation point may be configured to be evaluated periodically by the PE Scheduler (time-based). Alternatively, it may be configured to be evaluated when an event is received on a specified trigger point (event-based).

1.1.2 PI Performance Equations Recalculator

The **Performance Equations** is designed to adjust values of Performance Equation points automatically. The adjustment occurs when Archive values of points that are used in Performance Equation expressions are added, changed, or deleted. You can also use Recalculator as an offline utility to reprocess explicit periods of time for specific points.

1.1.3 PI PE Syntax and Functions Reference

PI Performance Equations allows you to easily implement sophisticated, real-time calculations, using data in the PI System. Calculations can include unit performance, real-time cost accounting, real-time yield accounting, batch summary, conversions and totalizations not performed by PI Totalizer, logical operations, and calculating aggregates.

This chapter provides comprehensive instructions for using Performance Equations syntax and functions.

1.1.4 PI Steam Functions Reference

The **PI Steam Functions** module is an extension to the PI Performance Equations Scheduler. Steam Functions provide a complete set of functions for deriving the thermodynamic properties of steam and water. PI Steam Functions support both English and SI units, and are based on the ASME (American Society of Mechanical Engineers) *Steam Tables*, *6th Ed*.

This chapter provides a comprehensive reference for setting up Steam calculations.

1.1.5 PI Batch Database

Most processes have repeatable time segments or stages. **PI Batch Database** technology maps process or manufacturing events to slices of time and data, and stores these in the PI Data Archive. Identifying these process stages and measuring their repeatability is the purpose of PI Batch. Building this fundamental association enables powerful data and process analysis for both traditional and non-traditional batch processes.

While industries such as chemical and pharmaceutical use PI Batch to track and analyze batches, it is also widely used in non-batch applications to identify and track process events. PI Batch tracks and stores batch and process based events hierarchically as *Batches*, *Unit Batches*, or *Sub Batches*.

PI Batch is used in conjunction with its companion client application PI BatchView, which allows you to view and compare events that have been collected by PI Batch and stored in the PI System.

1.1.6 PI Totalizer Subsystem

The **Totalizer Subsystem** performs common calculations such as totals, averages, minimum and maximum values, and standard deviations. The output of a calculation is stored in a PI point.

The main difference between a Performance Equations point and a Totalizer point is that Performance Equations are based on Archive events, while Totalizer results are based on Snapshot events.

PI Totals are the most accurate way to represent production summary data. Totalizers can be started and reset based on time and event, and ensure the highest accuracy in the calculation of flow volumes and other critical variables used to monitor product transfers or production performance. Totalizer is especially practical for totaling measurements or other process variables at the end of specific time periods, such as the end-of-day yields.

1.1.7 PI Alarm Subsystem

The **Alarm Subsystem** provides the capability to establish alarms for PI points. PI Alarm allows you to track, manage and acknowledge alarm conditions caused by processes that exceed user-specified parameters.

PI Alarm keeps a constant eye on process conditions. PI Alarm can monitor many variables such as temperatures, volumes, flow rates, product quality or raw material consumption. Alarms can be triggered by the duration of an event or deviation from norm.

PI Alarm will assess the condition as well as the priority of an event, as you define it. Depending on the longevity and/or severity of the event, it can notify specific personnel. PI Alarm includes client functionality through the PI-API to alert operators to selected alarms.

Data from PI Alarm are displayed in its companion client application, PI AlarmView. Alarm conditions are historized together with an acknowledgement status.

1.1.8 PI Real-Time SQC

PI Real-Time Statistical Quality Control (SQC) uses numerical methods to monitor the characteristics of a process, making sure they remain within pre-determined boundaries. When Real-Time SQC perceives an unacceptable deviation in a process, PI Real-Time SQC Alarms alert the appropriate personnel.

The PI Real-Time SQC component makes it easy to apply the Western Electric Pattern Tests to all of your process or laboratory data collected by the PI System. PI Real-Time SQC continually reviews any SQC tests in the PI System. It stores test results and a record of SQC control limits back into your PI System. The results are available for viewing and analysis via FactoryTalk Historian ProcessBook and the PI SQC Add-In.

The SQC Subsystem is a part of the PI Alarm Subsystem, which provides continual evaluation of SQC pattern tests and the management of alarms generated from them.

Chapter 2. PI PERFORMANCE EQUATIONS SCHEDULER

This chapter explains how to create calculated points using the **PI Performance Equations (PE) Scheduler**. The PE Scheduler allows you to implement sophisticated online calculations without having to program in high-level languages.

The PE Scheduler provides an equation syntax and library of built-in functions that allow you to perform a wide variety of calculations on PI data. (See Chapter 4: *PI Performance Equations Syntax and Functions Reference.*) Performance Equations can work with frequently-updating Snapshot and Archive values, whereas tools such as spreadsheets only have access to Archive data and limited access to Snapshot update values.

PE Scheduler allows you to easily implement sophisticated, real-time calculations, such as:

- Unit performance
- Real-time cost accounting
- Real-time yield accounting
- Grade-based costing
- Batch summary
- Conversions and totalizations not performed by PI Totalizer
- Logical operations
- Calculating aggregates

Each Performance Equation is associated with a PI point, and the calculation results are stored in the PI Archive as a *calculated point*, or *PE point*. You can configure a PE point to be evaluated periodically by the Performance Equation Scheduler on a *time-based* basis, or when an event is received on a specified trigger point, called *event-based* scheduling. The *Scheduling Method* is discussed in Section 2.5.

This chapter includes the following topics:

Typically, you use Performance Equations in one of two ways:

- ❑ To create *Calculated Points* **-** points that have the PE subsystem as their source. The PE Scheduler determines the value of these points by performing PE calculations specified during the creation of the calculated point.
- ❑ Programmatically - via the PI-SDK, FactoryTalk Historian DataLink or FactoryTalk Historian ProcessBook. Refer to product documentation for applicable Performance Equations instructions.

2.1 About Calculated Points

Calculated points perform calculations on one or more PI points. Calculated points are similar to other PI points, but they use the Performance Equation Subsystem as the point source. (The point source is specified in *pipeschd.bat* in the *\pi\bin* directory.) The PE Scheduler performs the calculations specified for the point at the scheduled time (based on "time" or "event") and sends the result to the Snapshot.

Note: In this guide, *Calculation Points* are also referred to as *PE Points* and these two terms are used interchangeably.

To create a calculated point, you put a calculation expression in the **Extended Descriptor (ExDesc)** attribute field and you set the point source to the point source for the PE subsystem. The value for the calculated point at any given time is the result of this calculation expression. The PE Scheduler calculates a new value for the point according to the schedule you define for it, either at regular intervals or using an event trigger.

You can use calculated points in other calculations, graph them in trend displays, or include them in reports, just like any other point.

2.2 About the PE Subsystem

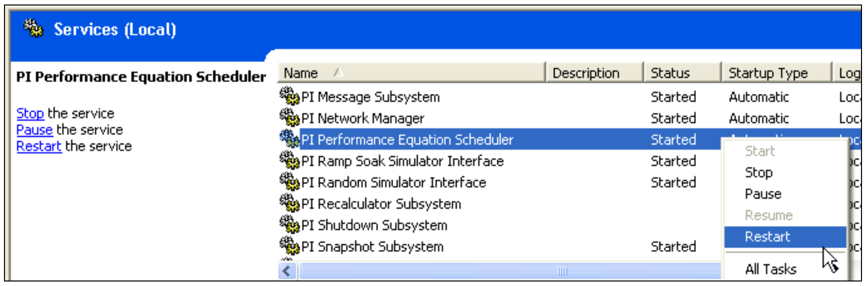
The PE Scheduler works a lot like an interface, except that it runs locally, on the PI Server. It has an associated *point source location* and *scan classes*. The PE Scheduler evaluates the calculation expression for each calculated point according to the schedule you configure for that point, and sends the resulting value and timestamp to the Snapshot. Calculated points are subject to exception and compression tests, just as other points are.

Like an interface, the PE Subsystem needs to be running in order to calculate data and send it to the Snapshot. The PE Scheduler executable is located in the *PI/bin* directory and is called *pipeschd.exe*.

2.2.1 Start and Stop the PE Subsystem

The PI startup script starts the PE subsystem, along with the other PI Server subsystems. Like other PI subsystems, if your PI Server is Windows-based it’s a good idea to run the PE subsystem as a Windows service, so that it can run in the background, independent of any particular login session.

On Windows computers, the PE subsystem typically runs as a Windows service and you manage it as you would any other Windows Service. The PE Scheduler service is called the **PI Performance Equation Scheduler**. Open the Services control panel (**Control Panel > Administrative Tools > Services)**, right-click on the **PI Performance Equation Scheduler** service, then start, stop or restart the service.



Services (Local)

PI Performance Equation Scheduler

Stop the service
Pause the service
Restart the service

Name	Description	Status	Startup Type	Log
PI Message Subsystem		Started	Automatic	Loc
PI Network Manager		Started	Automatic	Loc
PI Performance Equation Scheduler		Started		oc
PI Ramp Soak Simulator Interface		Started		oc
PI Random Simulator Interface		Started		oc
PI Recalculator Subsystem				oc
PI Shutdown Subsystem				oc
PI Snapshot Subsystem		Started		oc

On UNIX, use the `ps` command to determine the process ID, and then use the `kill` command to stop the process:

```
$ ps -aef | grep pipeschd
piadmin 25688 1 1 Sep 14 ? 35:22 ./pipeschd /Q /ps=C /ec=24 /f=00:
$ kill -2 25688
```

2.3 Procedure to Create Calculated Points

To perform a calculation on PI data, you must create a PI point; put your calculation expression in the **ExDesc** attribute field; set the **PointSource** attribute to the point source specified in the *pipschd.bat* file; and set **location4** to the **Scan** class.

Follow these steps to perform a calculation on PI data:

1. Determine the scan classes and point source. (See *Determine Scan Classes and Point Source* on page 8.)
2. Choose a scheduling method. You can use either *clock-based scheduling* or *event-based scheduling*. (See *Choose a Scheduling Method* on page 11.)
3. Create the point, and set the required attributes. (See *Set Attributes for Calculated Points* on page 12.) Create a new PI point using the PI tool of your choice, such as **PI Tag Configurator** or **piconfig**. Set the required attributes as follows:
   - **PointSource**: Set **PointSource** to the point source location specified in *pipeschd.bat* on Windows, or *pipeschd.sh* on UNIX. The default point source location for calculated points is the ASCII character, **C**, but you can edit *pipeschd.bat* to use another single alphanumeric ASCII character.
   - **Location4**: If you're using clock-based scheduling for this calculation, put the appropriate scan class in **location4**. The available scan classes are listed in *pipeschd.bat*. If you're using event-based scheduling, leave **location4** blank.

- **ExDesc**: Put your calculation expression in the **Extended Descriptor** (**ExDesc**) attribute field. The exact **ExDesc** expression depends on what type of scheduling (clock- or event-based) the point uses.
- **PointClass**: classic

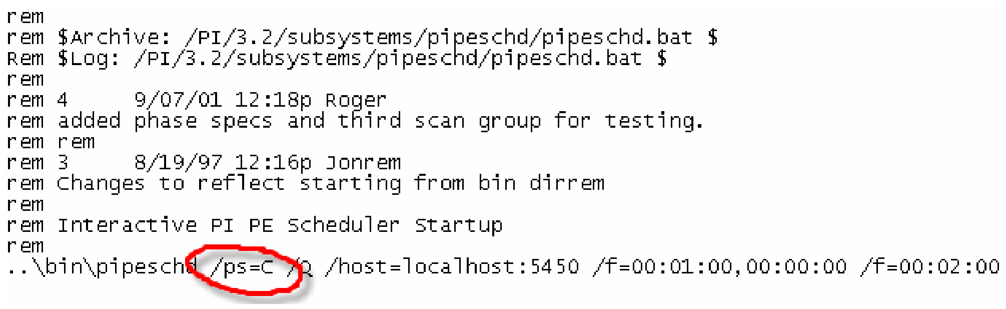
4. Test the PI PE calculation expression. You can use the **pipetest** utility to check whether an equation is syntactically correct. (See *Run the pipetest Utility*, on page 65.)

2.4 Determine Scan Classes and Point Source

PE Points use the scan classes and point source configuration for the PE Subsystem defined in *pipeschd.bat* on Windows, and *pipeschd.sh* on UNIX. By default, *pipeschd.bat* is located in the directory *PI\bin* where *PI* is the path to the main directory in your PI installation.

2.4.1 Find the PE Subsystem Point Source

To find the correct PE point source location for your PE points, open *pipeschd.bat* with a text editor and look for the entry `/ps=` (see below, circled). The point source is usually **C**, but it can be any single alphanumeric character.



```
rem
rem $Archive: /PI/3.2/subsystems/pipeschd/pipeschd.bat $
Rem $Log: /PI/3.2/subsystems/pipeschd/pipeschd.bat $
rem
rem 4     9/07/01 12:18p Roger
rem added phase specs and third scan group for testing.
rem rem
rem 3     8/19/97 12:16p Jonrem
rem Changes to reflect starting from bin dirrem
rem
rem Interactive PI PE Scheduler Startup
rem
..\bin\pipeschd /ps=C /Q /host=localhost:5450 /f=00:01:00,00:00:00 /f=00:02:00
```

Note: Make a note of the character. This is the value you must use for the **PointSource** attribute on all calculated points.

Change the Point Source

From a PI Administrator account, you can change the point source for PEs. Do not do this unless absolutely necessary. If you change the point source location, any existing calculated points will not work unless you change the **PointSource** attribute to match the new location.

To change the point source location for your calculated points, follow these steps:

1. Stop the PE Subsystem.
2. Change the point source to any single alphanumeric character. This is the value you must use for the **PointSource** attribute on all calculated points.
3. Make a note of the character and be sure to publish the new number to everyone who creates PE points for this server.

4. Edit all existing calculated points to reflect the new point source.
5. Start the PE Subsystem.

2.4.2 Specify the Optional Instance ID

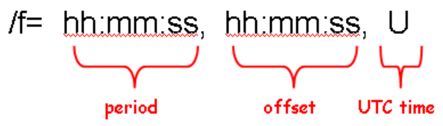
Within *pipeschd.bat*, you can optionally specify an *instance ID* **(/id=n)**. When the instance ID is specified, PI PE Scheduler only loads and calculates PE points with **Location1** attribute matching the **/id** value.

2.4.3 Find the PE Subsystem Scan Classes

All the scan classes available for a calculated point are listed in *pipeschd.bat*. If you don't see the scan class you need for a particular calculated point, you can add a new scan class (requires PI Administrator privileges.)

What is a Scan Class?

A *scan class* is a code that the PE Subsystem and other PI interfaces use to specify scheduling. Scan classes consist of a period, which specifies the interval between calculations and, optionally, an offset that specifies a start time for the calculations to begin – along with a code that specifies the UTC time to use for scheduling:



Scan Class Period

The **period** specifies the interval between calculations. The first two digits are the hours, the second two the minutes, and the third two the seconds. For example, the scan class can specify that the calculation take place every hour (01:00:00), every three minutes (00:03:00), every 52 seconds (00:00:52), etc.

Scan Class Offset

The **offset** specifies a start time for the calculation. The offset is optional. If no **offset** is included in the scan class, the first calculation takes place immediately. The **offset** is counted from midnight of the current day and, as with the period, the first two digits are the hours, the second two the minutes, and the third two the seconds. So, for example, the **offset** can specify that the first calculation occur at midnight (00:00:00), at 1AM (01:00:00), at 1PM (13:00:00), at 2:05PM (14:05:00), at 25 seconds past noon (12:00:25), etc.

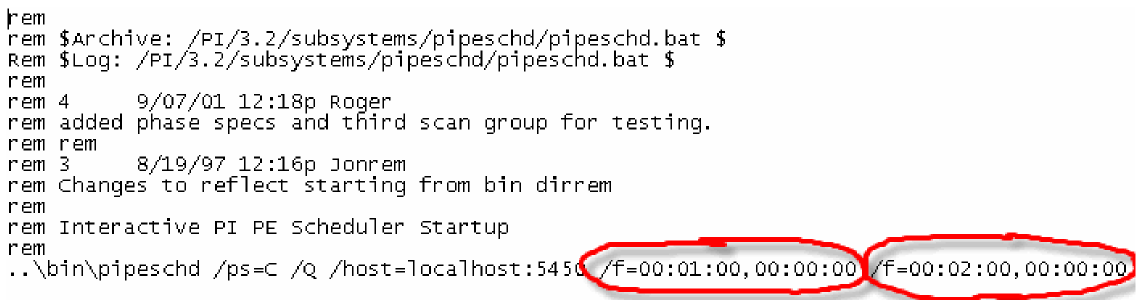
Scan Class UTC Time Flag

The **UTC time flag** specifies that the scheduling should sync with Universal Time Coordinate (UTC). The UTC time is optional. If a scan class has a frequency of more than an hour, make it a UTC scan class by adding a comma followed by a capital U: (/f=08:00:00,07:00:00,U). If you don't do this, then your scheduling might be inaccurate the next time there is a change to (or from) daylight savings time. UTC scan classes don't have this problem because they force the scan class scheduling to sync with UTC, rather than local time.

Find the PE Scan Classes

To see all currently available scan classes for your calculated points, follow these steps.

Open *pipeschd.bat* with a text editor and look for all the entries that begin with the characters *"backslash" f* **(/f)**. These are the scan classes. Choose the scan class that you want to use for the calculated point and set the **location4** attribute to the appropriate value.



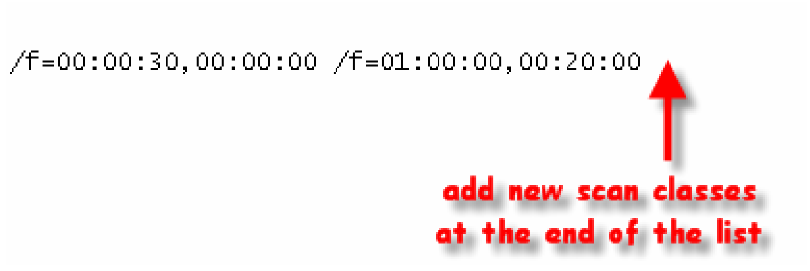
```
rem
rem $Archive: /PI/3.2/subsystems/pipeschd/pipeschd.bat $
Rem $Log: /PI/3.2/subsystems/pipeschd/pipeschd.bat $
rem
rem 4      9/07/01 12:18p Roger
rem added phase specs and third scan group for testing.
rem rem
rem 3      8/19/97 12:16p Jonrem
rem Changes to reflect starting from bin dirrem
rem
rem Interactive PI PE Scheduler Startup
rem
..\bin\pipeschd /ps=C /Q /host=localhost:5450 /f=00:01:00,00:00:00 /f=00:02:00,00:00:00
```

The position within the startup command line defines the scan classes; that is, the first **/f=** refers to scan class 1, the second **/f=** refers to scan class 2, etc. Simply add more **/f=** parameters to define more scan classes. The calculated point is assigned to a scan class using the **location4** attribute. For example, if **location4** is set to 2, the PE point will be evaluated every 2 minutes.

Add New Scan Classes

You can add new scan classes, if you have PI Administrator privileges. Stop the PE Subsystem before editing *pipeschd.bat*.

Add your new scan class at the end of the line containing scan classes, as the last scan class in the list. If you add a new scan class earlier in the list, you change the **location4** values for the existing scan classes. You can add as many new scan classes as you like to the end of the scan class list in *pipeschd.bat*.



```
/f=00:00:30,00:00:00 /f=01:00:00,00:20:00
```

Restart the PE Subsystem when you finish editing *pipeschd.bat*.

2.5 Choose a Scheduling Method

The attributes you set for a calculated point depend in part on the type of scheduling you use for the point. Each calculated point must use either clock (time-based) scheduling or event scheduling:

- ❑ **Clock Scheduling:** With clock scheduling, you use scan classes to schedule the calculation. The PE Scheduler calculates a new value for the point at the specified intervals, such as every hour, every five seconds, every 20 minutes, etc. You can optionally specify an initial start time for the calculation interval.

  For clock-scheduled points, define the PE calculation expression in the **ExDesc** attribute field and specify a scan class in the **location4** attribute field. (See *Set the ExDesc Attribute for Clock-Scheduled Points* and *Set the Location4 Attribute: Scan Class*, on page 13.)

- ❑ **Event Scheduling:** With event scheduling, you configure the calculation to occur when a specified point gets a new Snapshot value. For example, you might want a calculation performed whenever the point *ba:level.1* receives an update event.

  For event-scheduled points, put both the PE expression and the trigger tag name in the **ExDesc** attribute field (see *Set the ExDesc Attribute for Event-Scheduled Points* on page 15) and set the **location3** attribute field to specify the timestamp of the point. (See *Set the Location3 Attribute: Timestamp*, on page 13).

2.6 Set Attributes for Calculated Points

Table 2–1 lists the attributes that require a special setting for calculated points.

Table 2–1. Attributes that Require a Special Setting for Calculated Points

Attribute	**Requirement**
ptClassName	Classic.
PointSource	Set to value specified in *pipeschd.bat* file (or *pipeschd.sh* on UNIX). The default value is *C*.
location3	Output timestamp setting for event-based scheduling.
location4	Put the scan class here for clock-scheduled points. Leave blank for event-based points.
ExDesc	This is where you put your performance equation.
scan	Set the scan attribute to 1 (the default value)
shutdown	Marker inserted in Archive at shutdown.

All point attributes that are not listed in Table 2–1 work just the same as they do for other points. However, the following attributes do not apply to calculated points:

- **Location2**

- **Location5**
- **UserInt1**
- **UserInt2**
- **UserReal1**
- **UserReal2**
- **EventTag**
- **InstrumentTag**
- **SquareRoot**

2.6.1 Set the Point Source

The **PointSource** attribute for calculated points is defined in *pipeschd.bat*, discussed in *Find the PE Subsystem Point Source* on page 8. The default value is **C**; however you can change the defined value to any other single alphanumeric character by editing *pipeschd.bat*.

2.6.2 Set the Location3 Attribute: Timestamp

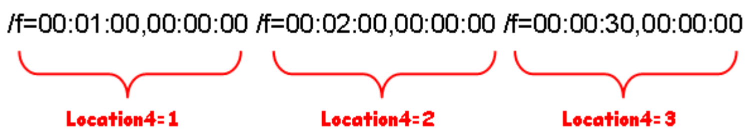
If the calculated point uses clock scheduling, do not set the **location3** attribute. Use the **location3** attribute for event-scheduled points, to specify how PI determines the timestamp for the calculated point. When **location3** is set to **0** (the default value), set the timestamps for the calculated point to the time when the expression is evaluated.

When **location3** is set to a non-zero value, the expression is evaluated at the timestamp of the triggering event and the timestamp of the resulting value is set to the timestamp of the triggering event.

2.6.3 Set the Location4 Attribute: Scan Class

If the calculated point uses event scheduling, do not set the **location4** attribute. If the point uses clock scheduling, set a value for **location4** to a positive non-zero integer that specifies a valid scan class.

You select a particular scan class for a calculated point by setting the value of the **location4** attribute for that point. All the scan classes available for a calculated point are listed in *pipeschd.bat* (called *pipeschd.sh* on UNIX computers). To select the first scan class in the list, set the **location4** attribute to **1**; to select the second scan class, set the **location4** attribute to **2**; etc.


/f=00:01:00,00:00:00 /f=00:02:00,00:00:00 /f=00:00:30,00:00:00
Location4=1
Location4=2
Location4=3

For an explanation of scan classes and how to configure them, see *Find the PE Subsystem Scan Classes*, on page 9.

2.6.4 Set the ExDesc Attribute: Calculation Expressions

For each calculated point, specify the PE calculation for the PE Scheduler to perform. PE calculation expressions use PE syntax and functions to define calculations, using data from other points. PE calculation expressions are similar to arithmetical expressions. You can use any of the standard arithmetic operators in a PE expression (such as **+**, **-**, or *****) to add the values of two points together, add a number to the value of a point, etc. You can also use *Performance Equation* functions and *Steam Table* functions in your PE calculation expressions (see Chapter 4, *PI Performance Equations Syntax and Functions Reference*, and Chapter 5, *PI Steam Functions* Reference.)

You define the PE calculation expression in the Extended Descriptor (**ExDesc**) attribute field, but the exact syntax you use depends on the type of scheduling you're using. For clock-scheduled points, you type only the PE calculation expression into the **ExDesc** attribute field, but for event scheduling you must also specify the point that acts as the event trigger.

Note: If the equation begins with a single quote (') and you are working with PI Tag Configurator, enclose the calculation expression in parentheses. Otherwise Excel will remove the single quote.

Set the ExDesc Attribute for Clock-Scheduled Points

For clock-scheduled points, the **ExDesc** field contains only the calculation expression itself. Several examples of simple calculation expressions are provided below.

The following example simply adds the current value of the *sinusoid* point to the current value of the *ba:level.1* point.

```
'sinusoid' + 'ba:level.1'
```

The following example takes the total time during the last hour that the *sinusoid* point had a value between 30 and 70.

```
timegt('sinusoid', '*-1h', '*', 30) - timegt('sinusoid', '*-1h', '*', 70)
```

For more examples of calculation expressions, see *Examples of Calculation Expressions* on page 16. For a complete reference on the PE syntax and functions, see Chapter 4, *PI Performance Equations* Syntax and Functions Reference.

Set the ExDesc Attribute for Event-Scheduled Points

For event-scheduled points, the syntax for the **ExDesc** attribute field is:

```
event = tagname, CalculationExpression
```

where *tagname* is the name of the point that triggers the calculation and *CalculationExpression* is the calculation expression that PE Scheduler uses to calculate the value for the point.

For example, to set up a one-hour average of the *sinusoid* point that triggers whenever *sinusoid* gets a new Snapshot value, use the following expression in the **ExDesc** attribute field:

```
event=sinusoid,TagAvg('sinusoid', '*-1h', '*')
```

The PE Scheduler uses the Snapshot value as the event trigger. This means that events that do not enter the Snapshot do not trigger the calculation. For example, an event that does not pass the exception reporting does not trigger a new calculation.

Character Limits on the Extended Descriptor Attribute

For both clock and event scheduling methods, there is a set limit on the number of characters that you can use for the extended descriptor. These limits depend on the tool you use to create your calculated point, rather than on the PE subsystem itself. In the PI TagConfigurator, the limit is 4096 characters.

Regardless of the limits on the extended descriptor, we recommend that you keep expressions less than 300 characters, if possible. Expressions that are much longer than that tax the system. If you need a longer expression, consider breaking down your equation into parts and creating calculated points to handle each of the parts. Be careful to schedule the calculations so that PE Scheduler can perform them in the correct order. See *Prevent Scheduling Errors* on page 19, for more information.

2.6.5 Set the Scan Attribute

Always set the scan attribute to **1** for calculated points. When the scan attribute is set to **0**, PI will not perform the calculations or generate any values for the point. Note that **1** is the default value for the scan attribute, so you can usually just leave this attribute as-is.

2.6.6 Set the Shutdown Attribute

Performance Equations are not calculated if the PI Server is not running, so you should always set the **Shutdown** attribute to **1**. When **Shutdown** is set to **1**, the system inserts a shutdown event with the timestamp of the PI Server shutdown. Note that the shutdown event is inserted only when the PI Server itself stops — it is not inserted if the PE Subsystem stops independently of the PI Server.

2.6.7 Examples of Calculation Expressions

This section provides some helpful hints and examples for writing Performance Equations.

Totalization of Digital Point Example

In this example, the goal is to obtain the total of the number of times a point goes into a digital state for the month. *Accumulator* is the PE point. *OnOffSwitch* is the digital point that uses a Digital State Set with two digital states: *ON* and *OFF*.

```
If day( ‘*’ )=1 and day(PrevEvent( ‘Accumulator’ ,  ‘*’ ))<>1 then 0 else if
PrevVal( ‘OnOffSwitch’ ,  ‘*’ ) <>  “ON”  and  ‘OnOffSwitch’  =  “ON”  then
'Accumulator’ +1 else NoOutput()
```

This performance equation checks whether it is the first of the month and whether the last event did not occur on the first of the month. If it is the first of the month, *Accumulator* resets. Otherwise if the previous value of *OnOffSwitch* is not the digital state *ON* and the current value is *ON*, then *Accumulator* increments.

TagTot Example

In this example, the goal is to use the **TagTot** function to obtain a total on a point that has engineering units other than the default per day. *RateTag* has engineering units of gallons per hour and the objective is to get the number of gallons for the previous day.

```
If PctGood( 'RateTag' ,  'y' ,  't' )>85 then TagTot( 'RateTag' ,  'y' ,
 't' )*24 else  "Bad Total"
```

First, the performance equation checks the percent of good values starting from the midnight yesterday to the midnight of the current day. If the percentage is greater than 85, then a total of *RateTag* is calculated for that given period. The total is multiplied by 24 hours per day to obtain the units of gallons. If the percentage is less than or equal to 85, the digital state of **Bad Total** is given. In this example, although the *RateTag* would have an integer or *real point type*, digital states only in the SYSTEM Digital State Set are allowed.

TagMax vs. Max Example

In this example, the objective is to obtain the maximum of a point for the month. One method for doing this is the **TagMax** function, shown in the next paragraph. An alternative method, also shown below, uses the **Max** function. Here's the calculation expression with **TagMax**:

```
If Day( '*' )=1 and Day(PrevEvent( 'RateTag' ,  '*' ))<>1 then
TagMax( 'RateTag' , Bom(PrevEvent( 'RateTag' ,  '*' )), Bom( '*' )- '+1s' )
else NoOutput()
```

The performance equation first checks that it is the beginning of the month and then finds the maximum of **RateTag** from the prior month up to one second before the beginning of the current month. Notice that the beginning of the month function **Bom** was not used to check for the first of the month. The following expression:

```
Bom( '*' )=  '*'
```

is not as accurate as the previous expression because the current time of the scan might not exactly equal the beginning of the month. Also the **TagMax** function may use too many resources accessing the Archive for data of the previous month and slow down the system.

Here's the calculation expression with **Max**:

```
If Day( '*' )=1 and Day(PrevEvent( 'RateTag' ,  '*' ))<>1 then
Max(TagZero( 'RateTag' ),  'RateTag' ) else Max( 'RateTag' ,  'MaxTag' )
```

This expression has the tagname of **MaxTag** and compares the point to be maximized **RateTag** to the current maximum in **MaxTag**. If the current time is the first of the month, **MaxTag** is reset by comparing the maximum between the current value of **RateTag** to the tagzero of **RateTag**. This version of obtaining a maximum makes only one Archive call as opposed to Archive calls to obtain one month of data.

2.7 Tips and Troubleshooting

This section contains the following topics:

- ❑ *Tips for Creating Calculated Points,* on page 18

- ❑ *Common Performance Equation Problems and Errors*, on page 18
- ❑ *Prevent Scheduling Errors*, on page 19
- ❑ *Prevent Skipped Calculations*, on page 19
- ❑ *When Data Types Don't Match*, on page 20

2.7.1 Tips for Creating Calculated Points

To avoid problems with your Performance Equations, follow these guidelines:

- ❑ Use the **BadVal** function to check to see if the value to be evaluated is bad, before carrying out a calculation.
- ❑ Use the **PctGood** function to check if the amount of data that is good, is within an acceptable level.
- ❑ Remember to escape anything that requires double quotes, with “double quotes”.
- ❑ Cascade calculations that occur in the same scan class by using offsets.
- ❑ Don’t use ambiguous timestamps.
- ❑ Set the **Step** attribute to **1** if you wish to have your data tracked as a stair-step instead of straight-line interpolation.

2.7.2 Common Performance Equation Problems and Errors

If you do not see results for a PE point, check the following:

- ❑ Make sure the PE Subsystem is running.
- ❑ Use the **pipetest** utility to check your equation syntax. Check your equation for any of the following common errors:
  - Are all PI times and tag names enclosed in single quotes?
  - Did you spell your tag names and function names correctly?
  - If the equation text begins with the single quote character (`'`), did you enclose the entire string in parentheses? Excel removes the leading single quote.
  - Are you (not) using tag names that are also valid PI time expressions?
- ❑ Check the log file *pipc.log* in the *pipc\dat* directory for Windows (*pipeschd.log* in *pi\log* for UNIX) to see if there are errors in the equations during compilation. In the log file, *error at offset x* indicates that a syntax error **x** characters from the beginning of the equation has been detected.

When the Performance Equation evaluator cannot perform a calculation during runtime, it returns the error value **Calc Failed**. This means that the PE point has the correct equation syntax and is running. Some possible causes are:

- ❑ Source tags have unexpected values. For example, the expression `'sinusoid' + 12` will result in **Calc Failed** if the value for the source tag *sinusoid* equals the digital state **Shutdown**.

- The source data do not meet the minimum **pctgood** value in a summary calculation. For example, `TagAvg('sinusoid', '*-1h', '*', 80)` will result in **Calc Failed** if less than 80% of the values for *sinusoid* are good for the last hour.
- Runtime data type conversion fails. For example, suppose the PE point is a digital point and has the expression `'StringTag'`. If the string source tag *StringTag* has a string that cannot be converted to a digital state either in the PE point's digital state set or the System Digital State Set, then the result will be **Calc Failed**.

2.7.3 Prevent Scheduling Errors

If two clock-scheduled calculations are evaluated at the same period and offset, there is no way to determine which calculation should be performed first. If your calculated point references other calculated points, you need to use an appropriate scan class offset to force PI to evaluate the calculations in a specific sequence.

For example, suppose points *A*, *B*, and *C* all represent calculated points. Point *C* is calculated as *A*/*B* (*A* divided by *B*), so point *C* should be calculated after points *A* and *B* are calculated. If all three points have the same scan class, there is nothing that ensures that points *A* and *B* will be calculated before point *C* is calculated. To trigger the point *C* calculation to take place after the point *A* and point *B* calculations, you can use a scan class with a slight offset for point *C*. For example, if you use the following scan classes for points *A*, *B* and *C*, then PI will calculate point *A* and *B* every hour on the hour, and then calculate point *C* a second later.

Point	Scan class
A	`/f=01:00:00,01:00:00`
B	`/f=01:00:00,01:00:00`
C	`/f=01:00:00,01:00:01`

2.7.4 Prevent Skipped Calculations

Although the only limit on the number of performance equations on a PI Server is the number points available to the PI Server, there are practical limits on the performance of PE Scheduler. It is possible for the PE subsystem to get overloaded.

If a scan class is more than one scan period behind, it will skip the calculation in order to catch up.

To see whether the PE Subsystem is skipping calculations, look at the *pipc.log* file located in the *pipc\dat directory* (or *pi\log\pipeschd.log* on UNIX).

If you find that the PE Subsystem is skipping calculations, we recommend you use the **Offset** attribute to stagger the calculation times so that large groups of calculations are not scheduled for the same time. For example, by using offset times of 10 seconds, 20 seconds, 30 seconds, and so forth, you can divide a set of five-minute calculations into thirty sub-groups to even out the system loading.

2.7.5 When Data Types Don't Match

When a calculated point's type does not match the type of the calculation, PE Scheduler converts the data type of the result to the data type of the calculated point (unless the result is a digital state).

In other words, for string points, the PE subsystem converts the calculation result into a string. For digital points, PE subsystem first converts the result into a digital state within the digital state set for the calculated point. If this initial conversion is not successful, the PE Subsystem converts the result into a digital state within the System Digital State Set. For numeric points, the PE subsystem converts the calculation result to an appropriate numeric value, such as integer or float.

If the data type conversion fails (for example, it is not possible to convert the string “ABC” into a numeric value), then the calculation expression returns the digital state **Calc Failed**.

Chapter 3. PI PERFORMANCE EQUATIONS RECALCULATOR

The **PI Performance Equations (PE) Recalculator** is designed to adjust values of PE points automatically. The adjustment occurs when Archive values of points that are used in PE expressions are added, changed, or deleted. You can also use PE Recalculator as an offline utility to reprocess explicit periods of time for specific points.

This chapter includes the following topics:

3.1 Recalculator Overview

The PE Recalculator automatically adjusts values of PE points when values of points used in PE expressions are added, changed, or deleted. Delayed or out-of-order Snapshot events can also trigger recalculations.

Note: PE evaluations based on new Snapshot values are performed by the PE Scheduler as described in *Choose a Scheduling Method* on page 11. Recalculator covers all times before the Snapshot value of a PE point.

The Recalculator supports multi-level (but not recursive) dependencies and takes into account the resulting time dependency. Explicit time dependency expressions and time-related functions are supported as well. Some point attributes of the dependent PE point and the source points are considered.

Like other standard PI subsystems, Recalculator runs on a PI Server Home Node, either as a service, or manually as a console application. When Recalculator is started as a service, messages are sent to the PI Message Subsystem and additional debug output may be sent to a log file. When Recalculator is started as an application, messages are written to the console.

There are several limitations and side effects to keep in mind, due to compression and other factors, which are described in this chapter.

It is important to realize that recalculation "is expensive" as it bypasses exception reporting, may retrieve a lot of Archive data for many tags, and may generate many out-of-order events. All of these factors place a significant overhead on a PI Server. In addition to these considerations, check that any affected Archive file contains the point definitions and has sufficient space.

3.1.1 Glossary of Recalculation Terms

The following recalculation terms are used in this chapter.

Table 3–1. Glossary of Recalculation Terms

Name	**Description**
Dependent Point	A PI point that normally receives its values from the PE Scheduler when it evaluates an expression. These are also the points that are modified by the PE Recalculator when necessary.
Source Point	A PI point whose tag appears in a PE expression. In general, additions, changes, and deletions of source point values trigger recalculations.
Absolute Timestamp	A date/time expression that evaluates to the same time regardless of the time of the calculation. Examples include '10-oct-99' and '01-jan-70.'
Relative Timestamp	A date/time expression defining an offset from the actual time of the calculation. Examples include '+7h' and '-30d'. Note: One exception to this rule occurs when a Relative Timestamp appears as one of the two time arguments of a PI PE Archive retrieval function. If the other time is an absolute time, it becomes the basis time.
Basis Time	The time to which the offset defined by the *Relative Timestamp* is applied. When evaluating PE point values, the PE Recalculator will determine the basis time of the dependent values to be corrected and will apply the offsets from that basis.
Combination Time	A date/time expression consisting of an absolute timestamp and a relative timestamp as an offset. Examples include 'T+7h.'
Performance Equation Point	Same as *Dependent Point.*

Name	Description
Inversion	Changing the sign of a *Relative Timestamp* offset in order to define the period of time requiring recalculation. For example, if an expression reads, `"TagVal('input', '*-1h')"`, then PE point values up to one hour after an `'input'` event, or `'*+1h'`, must be recalculated.

3.2 PE Recalculator Functionality

When the PE Scheduler starts, it finds the PE points by scanning the PI Point Database for points with a specific point source, usually **C**.

Since the Recalculator sends events to the same PE points, it also scans the PI Point Database on startup, scanning for the same point source. By default, all PE points are considered for recalculation. If you wish, you may assign values to the **Location1** attribute of any PE point. The Recalculator can be configured to consider only points with a specific **Location1** parameter value.

The PE Scheduler will sign up for exceptions for any event trigger points. The Recalculator signs up for Archive events of all source points. The reason for the difference is that the Recalculator must be aware of any changes to any source point, not just event trigger points.

Only events that were not handled in time by the PE Scheduler are considered by the Recalculator. The timestamp of the source point event has to be older than the Snapshot time of the corresponding dependent PE point. The delay caused by the normal scan cycle does not trigger a recalculation.

Recalculation is done in two steps: For a source point event, first the affected PE point periods are found. Then, all these periods are processed for existing events to replace, and new events to insert. When inserting new events, their timestamps are derived from the timestamps of the source events.

Generally, there are three main questions:

- ❑ Which dependent PE points are affected?
- ❑ What time range has to be recalculated?
- ❑ How do other point attributes influence the recalculation?

These questions are discussed here and lead to a definition of different recalculation types. The recalculation types are described in a subsequent section.

3.2.1 Point Dependency Considerations

Generally, all tags used in the **ExDesc** field of a PE point in the PI Point Database are classified as source tags. These tags may be used in arithmetic expressions or as function parameters.

With the PI Server, it is possible to dynamically construct tags by concatenating string constants and values of one or more string points. The Recalculator is unable to process expressions that use this construct.

3.2.2 Time Range Considerations

Time ranges are defined by two timestamp expressions, usually passed as two arguments to the same PI PE function. Examples include **TagAvg** and **FindEq**. It occurs frequently that the timestamp expression '*****' is used. This is basis time for the calculation. For PE points evaluated normally by the PE Scheduler, the basis time is the current time except for an event-based PE point with **Location3** set to one. In that case, the basis time is the trigger event time. For the Recalculator, the basis time is in general the time of the new or changed source point event.

Any new event for the source point not handled by the PE Scheduler will cause a series of Archive events within the time range to be recalculated. The Recalculator must determine the start and end time of the affected time range.

Time calculations resulting in a timestamp or in an interval can be *inverted* if no absolute timestamps are involved. For example, consider a PE expression such as:

```
TagVal('input_pt', '*-1h')
```

If **'input_pt'** receives a new event earlier than the Snapshot time of the PE point, its timestamp is the basis time for recalculation. PE point events one hour after the **'input_pt'** event are affected. The minus sign is effectively inverted to a plus sign, as in `'*+1h'`**.**

Inversion also applies to time periods:

```
TagAvg('input_pt', '*', *-1h')
```

If a new event for **'input_pt'** is received, its timestamp is the basis time, or **'*'** in the expression. The new event affects the period `'*'` until `'*+1h'`.

If there are absolute times in an expression, it would mean that all PE point values would have to be recalculated. In other words, if an absolute time appears in an expression, all values after the timestamp of a new or changed event would need to be recalculated.

Time Range Examples

Tag	Expression
TimeCalc	`If BadVal('xsource','*-8H') then NoOutput() else Tagval('xsource','*-8H')`
CalcInt	`TagAvg('xsource','*-1h','*')`
AllCalc	`If BadVal('xsource') then Tagval('xsource','12-Jan-98 08:00' ) else 'xsource'`

TimeCalc: In the first example, if the value of **xsource** at 12-Jan-98 12:12:00 is changed, **TimeCalc** at 12-Jan-98 20:12:00 (8 hours later) has to be recalculated.

CalcInt: In the second example, the expression has to be recalculated for a 1-hour period past the time of a modification of **xsource**.

AllCalc: If the value of xsource at 12-Jan-98 08:00:00 were to be modified, then all values of the PE point after 12-Jan-98 08:00 would have to be recalculated whenever **xsource** is in a bad state.

3.2.3 Clock-Scheduling vs. Event-based Scheduling

If a PE point is clock-scheduled, a new source point event causes a search for existing PE point events up to one scan cycle after the source point event to take place. The time between calculations depends on the **Location4** attribute indicating the scan cycle defined in *pipeschd.bat.*. If there is no Archive event found in the searched period, a new one is created.

In event-based calculations, no scan classes are defined and **Location4** is ignored. If an event within a default period (10 sec) does not exist, a new one is created for simple dependencies.

3.2.4 Step Point Attribute

The *Step* attribute defines how a trend of data values appears between events stored by PI Server.

If **Step = 0**, data values are linearly interpolated between archived events for numeric points. Changing a source point value implies changing all archived PE point values between the previous and the next event of the source point.

If **Step = 1**, data values value between archived events are considered to be the same as the previous event. Changing a source point event influences a dependent PE point between the modified source point event and its next event. Data values prior to the modified event are not affected.

3.2.5 Compression / Exception

Because of exception reporting and compression, not every original PE point event is found in the PI Data Archive. The Recalculator cannot simulate the original exception or compression algorithm because it no longer has the original Snapshot values. It recalculates for all events of all source points, plus all existing PE point events, plus start/end of the affected period.

There might be more Archive events after a recalculation. This might fail or take rather long when performed on Archive periods that are no longer covered by the primary Archive file.

3.2.6 Scan and Archiving Attributes

Set the **Scan** attribute to **0** to turn off the PE Scheduler. The **Archiving** flag causes no events to be sent to the PI Data Archive for storage.

If the **Scan** attribute is set to **0**, then there are no values generated by the PE Scheduler *at some time in the past.* If **Archiving** is **0**, values generated by the PE Scheduler are not stored. The times at which the **Scan** and **Archiving** attributes were edited are not recorded.

Note: The ability to recalculate events may be affected if either of these attributes are set to **0**. It is impossible to accurately correct history if there were no original events to adjust.

3.2.7 Location1

The **Location1** attribute may be used to exclude points or to schedule multiple instances of the Recalculator for performance reasons. This corresponds to the number in the **/in** startup parameter or in the **Instance** Registry setting. By default, all PE points are examined for recalculation.

3.3 Types of PE Point / Time Relationships

The types of the relationship between a PE point and time are summarized below, and explained in subsequent sections.

Table 3–2. Types of PE Point / Time Relationships

Relationship	**Short Description**
Simple	The expression consists of arithmetic operations and functions. There are no time parameters except **'*'**.
Multilevel Point Dependency	A PE point is source for another point. Archive modifications should trigger another recalculation.
Recursive	One of the source points in the expression is the PE point.
Relative Time Shift	The period to be recalculated is shifted according to the PE. No additional event for PE point is created at source event time.
Special Event	If the source event range covers the indicated event (*e.g.*, '**T'**), the period to be recalculated is determined according to the PE. Otherwise, the dependency relation can be ignored.
Time Period Reference	The PE contains Archive functions with start and end time. Start and End time have to be relative to a basis time (**'*'**, **'T'**, etc.)
Multilevel Time Dependency	The time parameter of an Archive function is the result of another function.
Absolute Time Reference	Automatic recalculation is not supported. This type of expression can only be recalculated manually.

3.3.1 Type 1: Simple Point Relationship

This type of recalculation covers all arithmetic functions and operations and all functions working on actual values only. Bad digital states are evaluated as with normal Performance Equations. Other errors will not change existing Archive events.

The question of the affected time range of the PE point has to be resolved even in this case.

Tag	**Expression**
TestCalc	`100 - 'sinusoid'`
T2Calc	`TagVal('TestCalc' , '*') * (('sinusoid'-50.0)/50.0)`

Finding Corresponding Timestamps

The main task is to find the timestamps for the modified source points that define the period of time that needs to be recalculated. In general, the timestamp of a PE point event is different from the timestamp of a source point. For the *Simple* recalculation type, the following situations depending on point attributes are considered:

- The PE always uses the values prior to the beginning of the scan cycle. In the example above, **T2Calc** is always one cycle behind **TestCalc**. The cycle is defined by the scan class of the PE points and by the scan parameters in *pipeschd.bat*.
- To insert a value, PE uses the system time, not the timestamp of an event trigger point by default.

As the calculated point may depend on more than one source point, the Archive is examined for events until the next event of the modified source point. All these events have to be recalculated. See Figure 3–1.

Inserting New Archive Events

When a new data event has been inserted for the source point, a new event of the dependent PE point may need to be inserted as well. The Recalculator searches the Archive for the next dependent PE point event after the trigger event. A point event found within one scan cycle will be modified; otherwise a new event for the dependent PE point will be inserted. An inserted event is set at the source event's timestamp +1 sec.

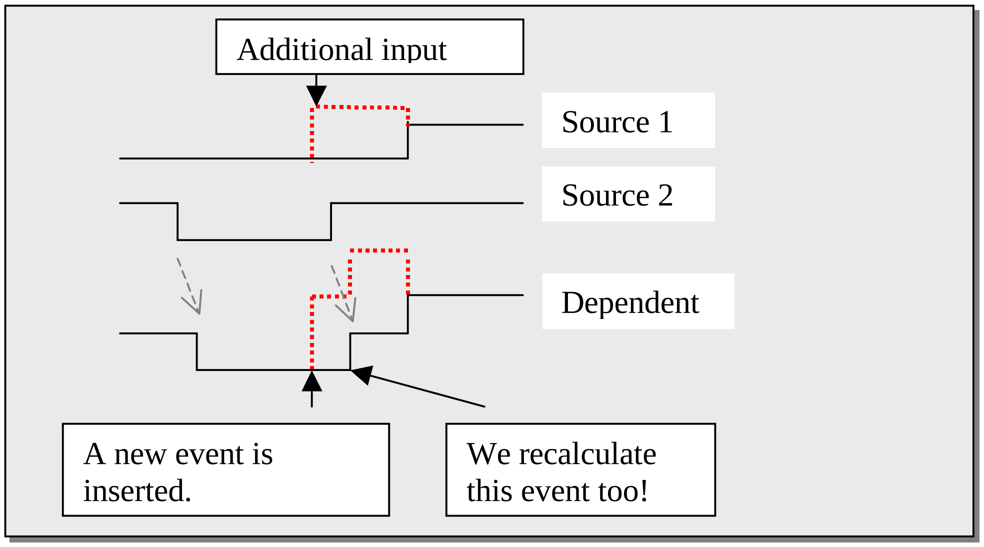


Figure 3–1. Recalculation Period on Type = Simple, Step=1, No Compression

Source Data with Step=0

If the **Step** attribute is **0**, a modification even affects the time prior to the changed value. The Recalculator examines the Archive in the range from the previous event to the next event of the modified source point. All events of the PE point in this period have to be recalculated.

Notice that a PE point event prior to the initiating source point may be modified. See the exclamation point (**!**) in Figure 3–2.

Note: With **Step = 0**, entering a single event into the Data Archive always affects the period between the previous and the next unmodified event. If you intend to enter a single peak or to mark the last "originally good value", you have to enter additional events.

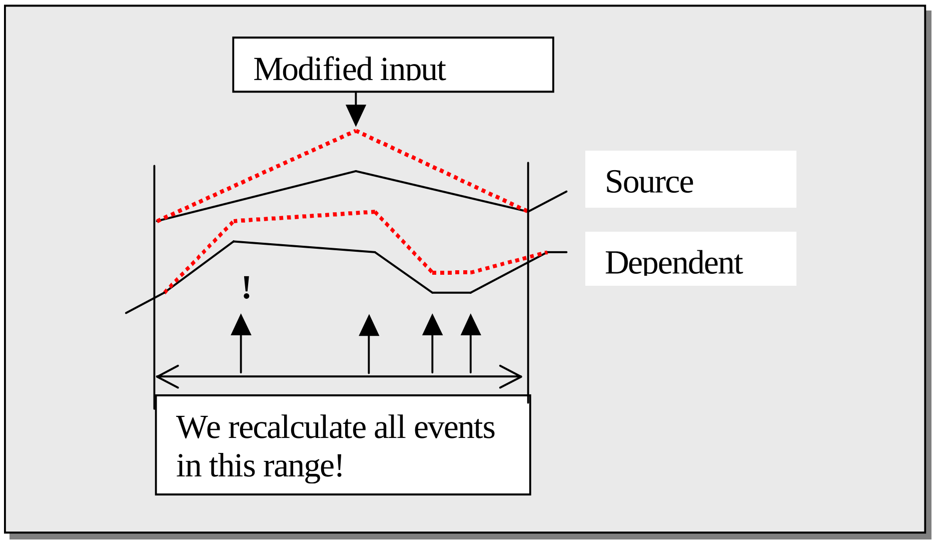


Figure 3–2. Recalculation Period on Type = Simple, Step=0, No Compression

Exception Reporting Algorithm

The PE Scheduler sends data to the Snapshot by exception to create events for dependent PE points. Depending on the exception parameters, this may result in suppressing new events if there are no relevant changes to the previous value. The compression algorithm minimizes the amount of available Archive events. In the following considerations, this is generally called *Compression*. The effect of the exception algorithm is not considered separately.

Compression on Calculated Values

There are no events in the Archive as long as the interpolated line between two events fits to the "real" values. The exception/compression algorithm cannot be simulated when the actual values are no longer available.

This implies extra considerations:

- ❑ The modified range of the PE point may be even larger than the affected source point range.
- ❑ Depending on the unmodified source point's event, a recalculation may yield different results compared to the straight-line result of recalculating only the given events of the PE point.

These considerations are shown in Figure 3–3.

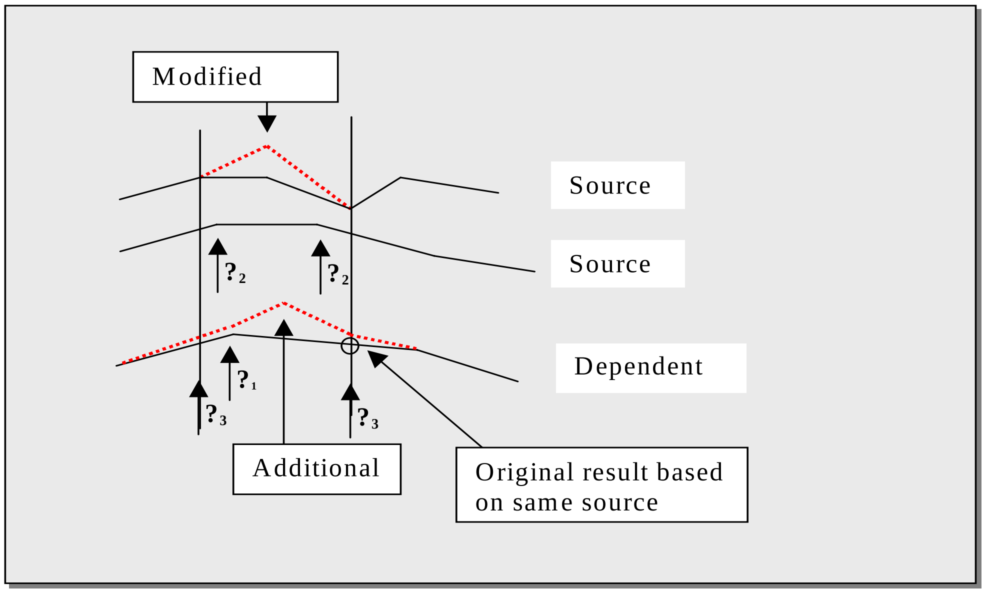


Figure 3–3. Recalculation Period on Type = Simple, Step=0, PE Point with Compression

Recalculator inserts extra events at the beginning and the end of the modified period (**?**$_3$) if necessary, so there are no new results without changed input. Due to the compression algorithms, the new values may still be slightly different from the original interpolated values.

Recalculator inserts extra events according to the modified source point event. Recalculator does not insert additional events due to unmodified source events (**?**$_2$).

Recalculator recalculates all existing dependent events in the time range affected by the modified source event (**?**$_1$). It does not simulate the exception reporting and compression algorithm.

Conditional Dependency

"If-then-else" constructions are evaluated only during recalculation. The point and time dependency is stated by parsing the expression in the **ExDesc** parameter for all cases on startup.

3.3.2 Type 2: Multi-level Dependency Point Relationship

A → B → C → …

(**"x → y"** means **x** influences **y**, **y** is dependent, and **x** is source)

Normally, no other special considerations are required if PE points are used as source points again. As new values of PE points are sent to the archive, there is an event on which the Recalculator has signed up, so the standard mechanism does all the work.

Recursive Point Relationship is checked for this dependency, too.

3.3.3 Type 3: Recursive Point Relationship

`A → B → A` or directly `A → A`

Such a recursive dependency is legal; you may use it for counting, etc. The result depends normally on "if-then-else" and on scan cycle parameters.

If a recursive dependency is detected, no automatic recalculation occurs. You may start a complete recalculation manually by using the **/execute** option and stating the desired time range. The result, however, depends on previous values and recalculations.

Examples

Tag	Expression
TestCalc	`If BadVal('TestCalc') then 0 else (if 'TestCalc' > 9999 then 0 else 'TestCalc' + 1)`
EvCount	`Event=CountMe, if BadVal('EvCount') then 0 else ('EvCount' + 1)`

TestCalc increments by one on every scan cycle and resets if the value is bad or is larger than 9999.

EvCount shows the number of events for the point *CountMe*.

The **BadVal()** constructions are required to quit any non-numeric initial state (**Shutdown, PtCreated**).

These recursive calculations are very sensitive to the previous state of the variable and may give different results when they have already been recalculated before. In example 2 (**EvCount**), there is no possibility to detect all events from the Archive. Additionally, they would generally result in a recalculation of the whole period up to now.

For these reasons, PE points depending recursively on themselves are generally excluded from recalculations.

There is no explicit limit on the number of levels to detect a recursive dependency.

`A → B → C →` … (any number of PE points) … `→ Z → A`

3.3.4 Type 4: Relative Point Relationship

`* - n X ; X ∈ {S, M, H, D }`

This relationship type is also known as a time shift relationship. For example:

```
Tagval('s_pt', '*-3h')
```

Inverting results in `* + nX`, so the Recalculator schedules recalculations at `'*+3h'`, when `'s_pt'` receives an Archive modification event.

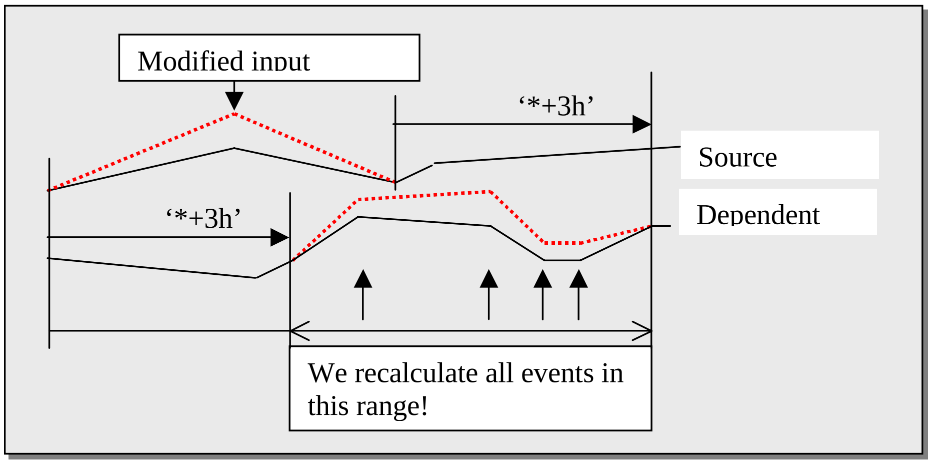


Figure 3–4. Type = Relative Time Shift

The algorithm described for type = *Simple* (without explicit time dependency) is applied to a period shifted by the indicated offset. Extra events are not created.

For the example here, this results in a recalculation period for the dependent PE point from `t1=(prevevent(source) + 3h)` to `t2=(nextevent(source) + 3h)`. See Figure 3–4.

Note: Future references `'*+nX'` are syntactically correct. Based on actual values, they result in **'No Data'** or **'Bad'**. On recalculation, they may result in numeric values. These are processed normally without error messages.

3.3.5 Type 5: Special Event Point Relationship

`B ± n X ; B ∈ {T, Y, 1 .. 31, Mon, Tue, … }`

This relationship type is also known as a *Relative Time Reference.*

The recalculation period may be one complete day relative to the given timestamp. The sign of the offset fraction is inverted. Note that only changes effective at the specified special time result in a recalculation.

Examples

Tag	Expression
BASEVAL	`TagVal( 'xsource' ,' T' )`
DIFFVAL	`'xsource' - TagVal('xsource','T')`

BASEVAL holds the midnight value of **xsource** of each day. If this value changes, all Archive values of **BASEVAL** for that day have to be modified. Every recalculation on this day yields the same new result. If there are changes of **xsource** at other timestamps, nothing is recalculated.

DIFFVAL has to be recalculated for the whole day, when **xsource** at midnight is modified. Other changes of **xsource** affect values of **DIFFVAL** only at the event timestamp of **xsource**, and the period of time between the previous and next events of **xsource**. The period of time between the new **xsource** event and the next **xsource** event would have to be recalculated if the **Step** point attribute is **1**. See Figure 3–5.

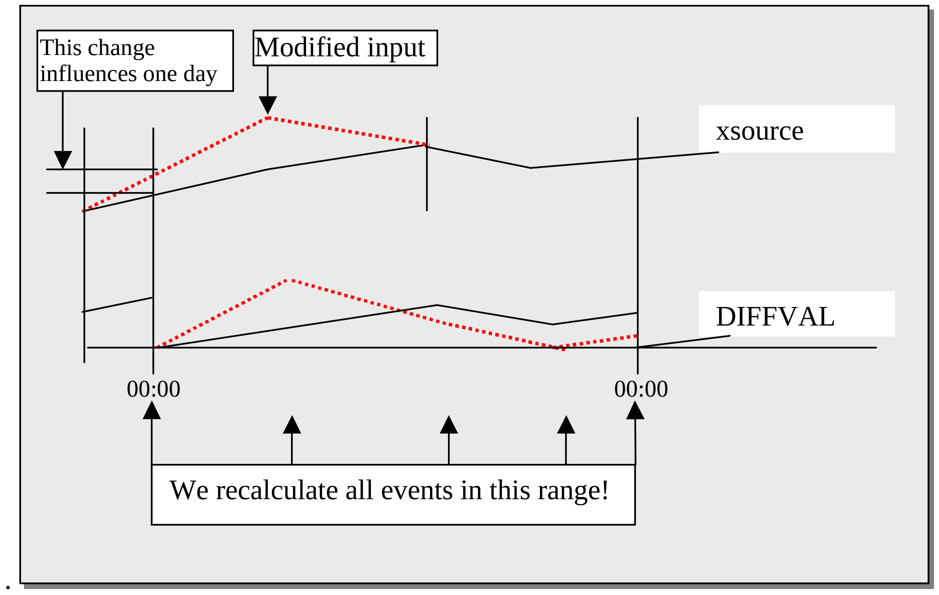

This change influences one day
Modified input
xsource
DIFFVAL
00:00
00:00
We recalculate all events in this range!

Figure 3–5. Type = Relative Time Reference

3.3.6 Type 6: Time Range Point Relationship

In the example below, any change of **xsource** affects the value of **YSUM** for the whole next day's period.

Tag	Expression
YSUM	`TagTot ('xsource','Y', 'T')`

This dependency type requires that both time parameters be relative to the calculation time. Expressions like **'-1h'** are valid as one of the time parameters. They are interpreted relative to the other.

3.3.7 Type 7: Multi-level Time Dependency

The time parameter of an archive function is the output of another function. For example, this expression finds the average value of *sinusoid* from the time yesterday at which the value of **'starttrigger'** was **50** to the current time:

```
TagAvg('sinusoid', FindEq('starttrigger','y','t',50), '*')
```

For recalculation purposes, the relation to the point *sinusoid* is considered a Type 1 (*Simple*) point relation. As the detection of the affected period does not consider the time dependency, not all possibly affected PE events are automatically recalculated. For example an edited value of *sinusoid* does not force the recalculation of the dependent PE point for the whole next day.

The calculation itself uses the time parameters correctly, so requesting a recalculation manually could be used as a workaround, if necessary.

3.3.8 Type 8: Absolute Time Reference Point Relationship

In the first example below, if the value of **xsource** at 01-Jan-98 is changed, the Recalculator would have to calculate everything from the timestamp stated in the PE expression up to now.

In the second example below, other changes of **xsource** affect the value of **DIFFVAL** only at the timestamp of **xsource** and the interval of time between the previous and next events of **xsource**.

Tag	Expression
BASEVAL	`TagVal('xsource','01-Jan-98')`
DIFFVAL	`'xsource' - TagVal('xsource','01-Jan-00')`

A change in an absolute timestamp in an expression does not cause recalculation of the whole Archive automatically. If this function is desired, you have to run the Recalculator as a console application, stating which point and time period has to be recalculated.

3.4 Special PE Time Functions

Some of the time functions in the PE library change the data type of their arguments, or extract information from timestamps. This category of time functions requires special consideration.

3.4.1 Modify a Timestamp

There are several time functions that modify a given timestamp including **Bod (), Bom (), Bonm (), Noon ()**. If the timestamp parameter is relative to calculation time (normally ***** or something based on it), the result refers to a special timestamp relative to the parameter. If this timestamp is modified, the Recalculator must recalculate a whole day or a whole month. This is similar to the **"B ± n X "** problem above (Type 4: *Relative Time Reference*). If the timestamp parameter is absolute, the result is an absolute timestamp; an automatic recalculation is suppressed (Type 8: *Absolute Time Reference*).

3.4.2 Parsetime Function

Parsetime () converts any string to a timestamp. If the input string is a constant and complete time expression, this has the same effect as a direct time parameter in single quotes. Here **Parsetime()** is not necessary (PI2 compatibility).

If the input string is an incomplete constant time expression, we have something relative to '*****' that evaluates similar to the previous functions.

If the input string is a variable, evaluation at compile time is impossible: Any automatic recalculation is suppressed.

3.4.3 Extract a Number from a Timestamp

Other functions extract a number out of a timestamp, including **Day (), DaySec (), Hour (), YearDay ()** .

These functions don't affect the recalculation period. If the results are used in other time expressions, they have the same effect as any other variables. The effects can only be evaluated during the recalculation process.

3.5 Examples of Archive Retrieval / Search Functions

A simple form of filter, assuming no compression, with scan-based values; this avoids recursive use of the PE point:

```
('xsource' + (PrevVal('xsource', '*')*9) / 10
```

Yesterday's average value:

```
TagAvg('xsource', 'Y', 'T')
```

Performs an integration (sum) on **xsource** since **tsource's** value first exceeded 10.0 yesterday; assuming **xsource** has a per day **EngUnit**:

```
TagTot(xsource, FindGt(tsource,'Y','T',10.0),'T')*24
```

These functions have several time parameters, defining a time range of Archive values to use. See the limits of the **Time Range Reference** type, earlier in this chapter.

PrevEvent(), or **NextEvent ()** return an absolute timestamp outside a given time range. This means theoretically that we cannot determine the reverse time. Practically they return the neighbor timestamp to the input timestamp. This is the same algorithm the recalculation performs anyway (if source point's **Step**=**0**). Therefore, no extra time dependency is evaluated. This applies also to the functions **PrevVal()** and **NextVal()**.

Note: Multilevel functions, such as **PrevVal('tagname'),** or **PrevEvent('tagname')-1s)** are calculated properly, but determining the affected recalculation period is not supported properly.

3.6 Recalculation Limitations

There are some restrictions and limitations on the Recalculator's ability to reprocess Performance Equation expressions. This section outlines these limitations.

3.6.1 Source Variables without Archive Values

If any of the source variables do not have Archive values, the recalculation results in **No Data.** This is not inserted into the Archive. If the original PE Scheduler calculation created Archive events, they are preserved. Recalculation of these equations is not performed.

3.6.2 Exact Simulation of the Original Scan Cycles

A point event found within one scan cycle will be modified; otherwise a new event for the dependent PE point will be inserted. An inserted event is set at the source event's timestamp +1 sec.

3.6.3 Modifications of the Performance Equation

When you modify a PE, recalculation is not initiated. Whenever recalculation does occur, the current expression is used. Previous expressions are not retained.

If you want to apply a new formula to past values, you must run the Recalculator explicitly on the desired point and period.

3.6.4 Archive and Time Functions

These generally result in recalculating some period between the input event and now. If these functions are used, they can cause a heavy amount of system load. As noted earlier, the recalculation of the whole Archive up to now because of absolute timestamps is suppressed on automatic recalculation.

Timestamps as a result of other embedded functions are not supported completely. If the timestamp expression cannot be evaluated at compile time (subsystem start), a dependency Type 1 (i.e. *Simple*) is assumed. This applies to expressions like:

```
TagVal('input_tag', PrevEvent('input_tag','*')-3600)
```

Not all affected PE point events might be found correctly. See the description of Type 7: *Multilevel Time Dependency* for another example.

It is syntactically correct to have relative time expressions referencing future values. PE Scheduler works on **No Data** and gets a **Calc Failed** result. If the Recalculator has to perform this calculation, this is not considered specially. A numerical value might be returned and stored in the archive. So there are different results without any change in the source data.

3.6.5 Unsupported Dynamic Functions

Recalculator does not support the following functions: **Arma (), Delay (), Impulse ()**, and **MedianFilt ()**.

3.6.6 Incomplete Timestamps

An incomplete absolute timestamp (e.g., **'25'** = midnight on the 25th of the actual month) refers to the actual time of compilation, which gives different results depending on the time of recalculation. For more details, see *Set the Location3 Attribute: Timestamp,* on page 13. Avoid these expressions.

The calculation is independent of the Archive time for which it will be used.

As incomplete timestamps are not absolute dates, they don't lead to a total recalculation of any dependent PE point. As they are not valid relative time expressions, inverting of time dependency is not performed.

3.6.7 Blob Support

Like the PE Scheduler, the Recalculator does not handle Blob type points.

3.7 Recalculator Point Configuration

Points used by the Recalculator are completely defined by the requirements of the PE subsystem. For more information regarding point configuration, see Chapter 2, *PI Performance* Equation.

In addition, the **Location1** parameter is used. The **Location1** value does not influence the operation of the PE Scheduler.

The Recalculator uses the **Point Name** and **Step** parameters from the source points. The Recalculator signs up for events of the source points, which may belong to any point source.

The Recalculator is scheduled with the point source class of PE points, and uses the following parameters of the dependent PE points: **Extended Descriptor (ExDesc), Point Source, Scan flag, Archiving flag, Location 4,** and **Point Type**, as described below.

3.7.1 Point Name

The point names for the PE point and the source points follow the normal PI point-naming conventions.

3.7.2 Extended Descriptor

The **Extended Descriptor** (**Exdesc**) contains the Performance Equation. It is analyzed to find source points and time expressions. It is calculated to get the new Archive values.

3.7.3 Point Source

All points defined in the PI Point Database for use with this module must share the point source of the PE subsystem. The **Point Source** is a single character. The standard for the PE Subsystem is **C**.

3.7.4 Scan

The **Scan** flag must be set to **1** for the PE Scheduler to work on this point. The Recalculator does not use or change the **Scan** flag.

3.7.5 Archiving

Archiving has to be set to **1** for the PE Scheduler to create Archive events of this point. The Recalculator does not use or change the **Archiving** flag, but the **Archiving** flag must be set to **1**.

3.7.6 Location1

By default, the Recalculator considers all PE points as candidates for recalculation and does not use this attribute value. However, if there is a parameter `"/IN=n"` where `n > 0)` in the startup file or an **Instance** value in the Registry, only points with the corresponding value in **Location1** are recalculated. This feature can be used to assign PE points into recalculation groups.

Note: It is possible to start multiple instances of the Recalculator. Every instance should be controlled by a different **/in** parameter.

3.7.7 Location4

This parameter specifies the scan class used by PE Scheduler.

3.7.8 PointType

As with the PE Scheduler, the point types **Int16**, **Int32**, **Float16**, **Float32**, **Float64**, **Digital** and **String** may be used.

3.7.9 Step

This value for the source point is used to determine if any event any time prior to the scheduled event must be examined. See *Point Dependency Considerations*, on page 24.

3.7.10 Other Attributes

The Recalculator does not use any other point attributes. They may be used by the PE Scheduler and retain their normal meanings for other operations.

The following attributes are not used:

- **Location2**
- **Location3**
- **Location5**
- **UserInt1**
- **UserInt2**
- **UserReal1**
- **UserReal2**
- **EventTag**
- **InstrumentTag**
- **Square root code**

3.8 Start Recalculator as a Service

The Recalculator can run as a Windows service, to automatically recalculate PE points that have source points that receive edits, deletions, or new delayed or out-of-order events.

The Recalculator module is normally installed automatically when the PI Server is installed. This section explains how to configure the PE Recalculator startup.

3.8.1 Configure Startup and Shutdown

The Recalculator is not set to start automatically upon system startup, even if you chose to have the PI Server start automatically. To start Recalculator automatically, edit the startup and shutdown scripts, and change the startup settings for the Recalculator service. You should do both since you may start the PI Server by starting the computer, or by running the startup script when the computer is already running.

Startup and Shutdown Scripts

Edit *\pi\adm\ pisrvsitestart.bat* and add the line:

```
net start pirecalc
```

Edit *\pi\adm\ pisrvsitestop.bat* and add the line:

```
net stop pirecalc
```

Edit *\pi\adm\ pisitestart.bat* and add the line:

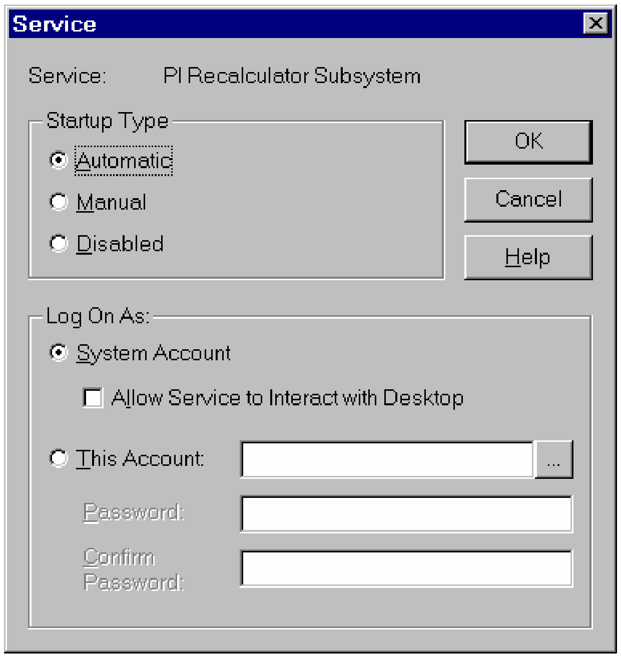
```
start "PI Recalculator Subsystem" /min cmd /k ..¥bin¥pirecalc.exe
```

Configure Automatic Startup

The Recalculator may be configured for automatic startup by either of two methods:

Using the Control Panel Services Applet

Open the **Control Panel** and start the **services** applet. Locate the **PI Recalculator Subsystem** in the list of services. Click the **Startup** button:


Service

Service: PI Recalculator Subsystem

Startup Type: [✓ Automatic] Manual Disabled

Log On As: [✓ System Account] This Account

☐ Allow Service to Interact with Desktop

This Account:

Password:

Confirm Password:

Click the **Automatic** radio button. The Recalculator will start automatically on your next reboot.

Using Recalculator Command-Line Arguments

The Recalculator executable supports command-line arguments that can be used to configure it to run as a Windows service.

Issue the following commands:

```
¥pi¥bin¥pirecalc -remove
¥pi¥bin¥pirecalc -install -auto –depend piarchss
-display ″PI Recalculator Subsystem″
```

3.8.2 Specify Options with a Startup Script File

When the Recalculator starts, it searches for *\pi\bin\pirecalc.bat*. This file may contain startup and debugging options, and can also be used as the startup file when starting the Recalculator manually. This file is not created automatically when the PI Server is installed or upgraded.

Note: When specifying file names in the script, be sure to use full path names.

You can also set some of these options by editing the Windows NT/2000 Registry. See Section 3.8.3, *Specify Options*, for details.

Recalculator Startup Options

Table 3–3 shows the command line switches for the PE Recalculator. These options are detailed after the table.

Some of these options are best used only when the Recalculator is started manually. See section 3.9, *Start Recalculator Manually*, for details.

Table 3–3. Recalculator Startup Options

Parameter	Description
/in=0	Interface number (corresponds to **Location1** point attribute). Omitting this parameter or a value of **0** means ignore **Location1** values.
/output=c:\….\pirecalc.log	Module-specific debug log file pathname. The default output is to the screen, if run as console application.
/ex[ecute]=tag,start[,stop][,TEST]	Recalculate a specific PE point and exit. This option can accept one timestamp to specify a point in time, or two timestamps to specify a range. Adding the word **'TEST'** causes the display of recalculated results with no storage in PI. Wildcard characters can be used in the tag. If present, an **/in=** startup parameter is checked, too. This option may only be used only in manual mode.
/deb=0	Debug level. See Table 3–4 for debug level options.

These parameters of *pipeschd.bat* are also read and interpreted:

/f=…	Scan Class frequency, optional use of multiple **/f=**… -no default values provided -
/ps=C	Point Source

Note: Command-line parameters are case-insensitive. You can use a leading **-** (hyphen) instead of **/** (forward slash) as well.

Command-Line Parameter Reference

/ps = C

Specifies the point source of the points on which the module will operate. This parameter is taken from the PE Scheduler startup script file, *pipeschd.bat*.

/in = 1

This parameter corresponds to the **Location1** attribute of a point. If you omit this parameter or set it to zero, all PE points, determined by the **/ps** parameter in *pipeschd.bat*, are checked for recalculations.

/f = 00:00:30[,00:00:00]

This parameter is set in *pipeschd.bat*. It defines the scan frequency for different scan classes. There is no limit on the number of scan classes. An offset may be added.

These parameters in combination with the **Location4** point attribute determine where an existing PE event is searched, or if a new event is created. The offset portion is not used.

/output = C:\Program Files\Rockwell Software\FactoryTalk Historian\Server\PI\log\pirecalc.log

The **/output** option causes Recalculator to generate debug output and error messages and send them to *C:\Program Files\Rockwell Software\FactoryTalk Historian\Server\PI\log\pirecalc.log*. If you omit this parameter and start Recalculator (manually) as a console application, output is sent to the console window.

If you start Recalculator as a service and the output parameter is missing, output goes to the PI System Message Log. Use the **PIGetMsg** utility or **PIHealthCheck** to retrieve this information. Debug messages are not sent to the PI System Message Log.

/execute = tagmask,starttime[,endtime][,TEST]

The **"/execute"** mode allows you to test or modify single Dependent PE points. Use this option only when you manually start the Recalculator.

For example: `"/execute=tag1,12-dec-98 15:00:00"` means, recalculate the PE point *tag1* at the given timestamp, then exit.

- For *starttime* and *endtime* the PI time syntax is allowed.

Note: Do not use quotes around *tagname* and *time*. If this option contains space characters (timestamps require a space between date and time), enclose the whole option in double quotes.

- Tagmask is searched among the tags with the same point source as stated in parameter **/ps** in *pipeschd.bat*.
- If a parameter **/in** is present, **Location1** is checked, too.
- Tagmask can contain wildcard characters ***** and **?**. All matching points are recalculated.
- If there are two time parameters, they define a time range to be recalculated.
- If there is a single time parameter and no event exists at that time - within limits given by the corresponding scan cycle, a new Archive event is created.
- If the additional sub-parameter **TEST** is applied, the results are not stored in the Data Archive but are printed only, according to the **/output** parameter.
- This mode doesn't work when run as a service.
- The option keyword may be abbreviated up to **"/ex=".**

/deb=0

The module is able to print additional debug information into the module-specific log file, depending on the debug level used. The amount of log information increases with the value. All information of lower levels is included.

Table 3–4 lists allowed debug values, and output.

Table 3–4. Recalculator Debug Levels and Output

Debug Level	**Output**
0	Start / Version / Number of points / Stop (As sent to Message Subsystem). Test mode results. Internal errors.
1	Additional information about module operation. Examples: startup parameter / defaults, results of dependency check on start/update.
2	Information about problems that will be handled by the module and will not cause data loss. Start of a recalculation period.
3	Display result dependency. More calculation period info.
4	Print out each calculation the program performs. Only for onsite test purposes. Use this mode if directed by OSI Tech Support. Log file might increase rather quickly.
5	More information than Level 4.
6 … 8	Additional internal debugging information
9	Maximum internal dump output.

3.8.3 Specify Options with the Windows Registry

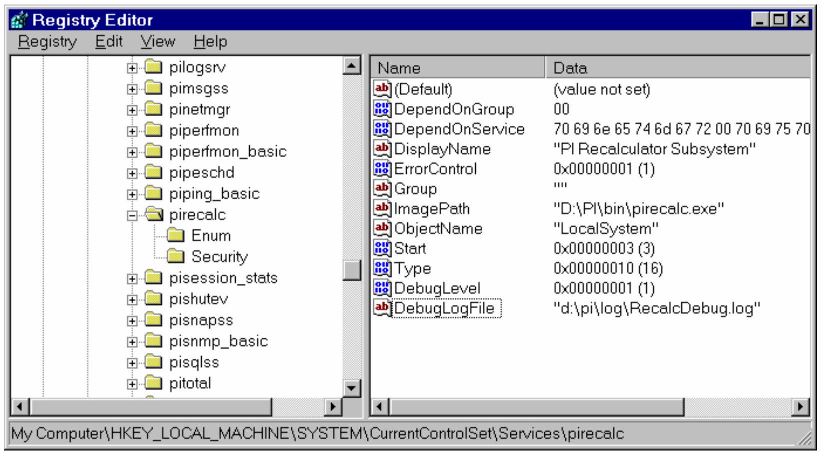
To control Recalculator when started as a service, you can specify startup options in the Windows NT/2000 Registry rather than creating a startup script file. Registry values are not created automatically when the PI Server is installed or upgraded. Use the RegEdit application to update the Registry.

You can include the following startup parameters in the Registry key **KLM\SYSTEM\CurrentControlSet\Services\pirecalc:**

Startup Parameter	**Registry Value Name**	**Data Type**
/output	`DebugLogFile`	`REG_SZ`
/deb	`DebugLevel`	`REG_DWORD`
/in	`Instance`	`REG_DWORD`

Note: Startup file settings located in *pi\bin\pirecalc.bat* have priority over Registry settings.

If there is no path information for *DebugLogFile*, the standard PI Server log directory *C:\Program Files\Rockwell Software\FactoryTalk Historian\Server\PI\log* is assumed.



Name	Data
(Default)	(value not set)
DependOnGroup	00
DependOnService	70 69 6e 65 74 6d 67 72 00 70 69 75 70
DisplayName	"PI Recalculator Subsystem"
ErrorControl	0x00000001 (1)
Group	""
ImagePath	"D:\PI\bin\pirecalc.exe"
ObjectName	"LocalSystem"
Start	0x00000003 (3)
Type	0x00000010 (16)
DebugLevel	0x00000001 (1)
DebugLogFile	"d:\pi\log\RecalcDebug.log"

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\pirecalc

The example above shows the Recalculator service configured for a debug level of **1**, with a debug log file specified.

A normal installation has neither Registry keys nor the *pirecalc.bat* file.

3.8.4 Run Multiple Instances

To recalculate several subsets of Performance Equation points you may wish to run multiple instances of **pirecalc**. Create a startup script for each instance, such as:

```
¥pi¥bin¥pirecalc.bat
¥pi¥bin¥pirecalc2.bat
¥pi¥bin¥pirecalc3.bat
```

They should differ by the **/in=#** startup parameter. If they are intended to start console applications, they may refer to the same executable, **\pi\bin\pirecalc.exe**. See Section 3.9, *Start Recalculator Manually*, for more information.

To run multiple instances of Recalculator, create copies of *pirecalc.exe* and rename them, for example:

```
¥pi¥bin¥pirecalc2.exe
¥pi¥bin¥pirecalc3.exe
```

Then install them as services by running

```
¥pi¥bin¥pirecalc2.exe -install -auto -depend piarchss
-display "PI Recalculator Subsystem (Subset 2)"
```

It is necessary that corresponding *.exe* and *.bat* files have the same base names and reside in the same directory *\pi\bin\*. You may change the display (icon) names. Rockwell Automation recommends you start them with "PI." They must be unique.

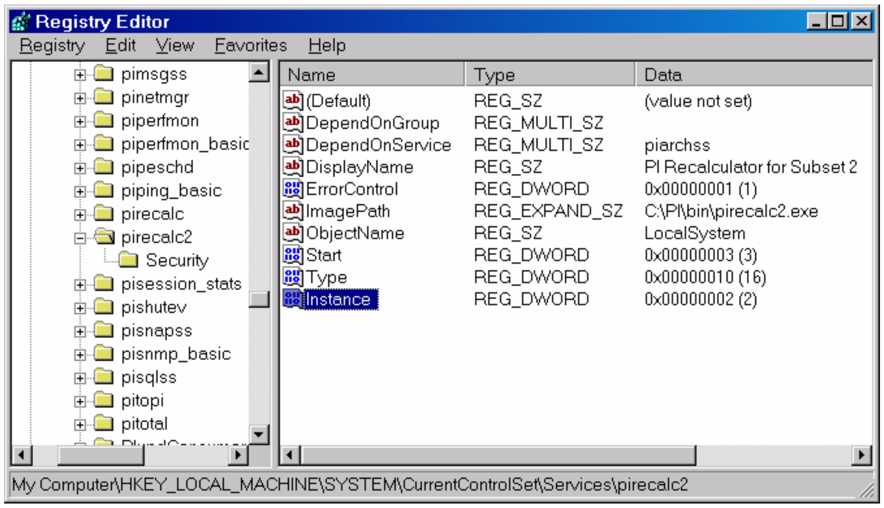
Edit the *start* and *shutdown* scripts: see the instructions in Section 2.2.1, *Start and Stop the PE Subsystem*, for more information. The notes there about automatic vs. manual service start apply to the additional instances as well.

To rename the originally installed instance, you have to remove the service first. Use the command:

```
¥pi¥bin¥pirecalc.exe -remove
```

And proceed as described above.

Alternatively to different *.bat* files, you may add the **Instance=#** Registry value as described in Section 3.8.3, *Specify Options*. The Registry key **pirecalc2** might look like this:



pimsgss
pinetmgr
piperfmon
piperfmon_basic
pipeschd
piping_basic
pirecalc
pirecalc2
Security
pisession_stats
pishutev
pisnapss
pisnmp_basic
pisqlss
pitopi
pitotal

Name	Type	Data
(Default)	REG_SZ	(value not set)
DependOnGroup	REG_MULTI_SZ	
DependOnService	REG_MULTI_SZ	piarchss
DisplayName	REG_SZ	PI Recalculator for Subset 2
ErrorControl	REG_DWORD	0x00000001 (1)
ImagePath	REG_EXPAND_SZ	C:\PI\bin\pirecalc2.exe
ObjectName	REG_SZ	LocalSystem
Start	REG_DWORD	0x00000003 (3)
Type	REG_DWORD	0x00000010 (16)
Instance	REG_DWORD	0x00000002 (2)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\pirecalc2

3.9 Start Recalculator Manually

You can start Recalculator as a console application instead of, or as well as, a Windows service. This is useful if you want to watch debug message output on the screen, or to reprocess Performance Equation expressions on demand.

3.9.1 Recalculator Startup Options

When started manually, the Recalculator interprets the same startup options as when run as a service. See *Recalculator Startup Options*, on page 41.

An example of a startup script is as follows:

```
pirecalc.exe /deb=1 /output=c:¥Program Files¥Rockwell Software¥FactoryTalk
Historian¥Server¥pi¥log¥pirecalc.out
```

You can start the Recalculator by running a batch file (such as *\pi\bin\pirecalc.bat*) containing these startup options.

You can recalculate a subset of PE points if they have the same **Location1** parameter. This is specified with the **/in** startup parameter. Different subsets can be processed in parallel by other instances of **pirecalc**, that are run as console or as a service. Running several console application instances in parallel simply requires different startup scripts, containing different **/in** parameters.

3.9.2 Manual Recalculations

This section describes recalculation operations that you can perform when you start Recalculator manually.

Recalculate a Single PE Point over a Period of Time

Use the **/execute** option with two timestamps. For example:

```
/execute=MyAvgTag,"01-dec-98","01-jan-99"
```

Recalculate a Single PE Point at a Specific Time

Use the **/execute** option with a single timestamp:

```
PIRECALC "/exec=petag,timestamp"
```

A new value is added if there is no event in the Archive at or directly (within one scan period) after the timestamp. For example:

```
/execute=MyOtherTag,"30-nov-98 23:59:59"
```

Test Recalculation over a Period of Time

Recalculator shows the expected value of a PE point by recalculating it, but does not store the result in the PI Data Archive:

```
pirecalc /exec=T2Calc,Y+7h,Y+15h,TEST
```

Recalculator shows **T2Calc's** values from yesterday at nine o'clock in the morning, to three o'clock in the afternoon. The Archive remains unmodified and might contain different values. **T2Calc** is assumed to be a dependent PE point.

Test Recalculation at a Specific Time

Recalculator shows the expected value of a PE point by recalculating it, but does not store the result in the PI Data Archive:

```
pirecalc /exec=T2Calc,Y+9h,TEST
```

The Recalculator shows **T2Calc's** value from yesterday morning at nine o'clock. The Archive remains unmodified and might contain a different value. **T2Calc** is assumed to be a dependent PE point.

3.10 Stop Recalculator

If you start Recalculator in single execution mode (that is, manually with the parameter **/execute**), the module stops itself when finished. If you start Recalculator as a console application, use the **<Ctrl>-C** or **<Ctrl>-<Break>** key command.

If you start Recalculator as service, you can stop it via the Control Panel, or with the command: (where *PI* is the full path of the PI directory)

```
PI¥bin¥pirecalc -stop
```

or

```
net stop pirecalc
```

3.11 Optimize Recalculator Performance

Recalculator is intended to operate as a background task on already existing Archive events. To limit the system load, recalculation is divided into several steps with decreasing priority:

- Find the PE point and period to be calculated, whenever triggered by a source event.
- Perform calculations of periods found in the previous step.
- Check for point attribute updates.

These steps are triggered in different cycles and are limited to a number of operations per cycle. You cannot change the parameters controlling this behavior.

To speed up the performance of the single execution mode, select an appropriate subset of dependent PE points using the point attribute **Location1** and the **/in=** startup parameter.

3.12 Error and Information Messages

You can view Recalculator system (log) messages in two locations.

- The standard PI System Message Log contains general information and error messages, such as: *subsystem start / stop*, *used point source character*, and *number of points handled*. Use **pigetmsg** or **PIHealthCheck** to read the Message Log.
- You can instruct Recalculator to generate a user-configurable message log file. Output messages to any filename with the **/output** startup parameter. (If you start Recalculator as an application, the output defaults to the console window.) Configure the error and debug information recorded with the **/deb=level** startup parameter. The log also records results if you use the **/execute=tagname,period,TEST** startup argument.

Chapter 4. PI Performance Equations Syntax and Functions Reference

The **Performance Equations (PE) Scheduler** allows you to easily implement sophisticated, real-time calculations, using data in the PI System. (See Chapter 2, *PI Performance Equations Scheduler*.)

The PE Scheduler includes an equation syntax and a library of built-in functions, which allow you to easily perform a wide variety of calculations on PI data. Typical calculations include unit performance, real-time cost accounting, real-time yield accounting, heat and material balances, batch summary, conversions and totalizations not performed by PI Totalizer, logical operations, and calculating aggregates.

This chapter provides comprehensive instructions for using Performance Equations syntax and functions, and includes the following topics:

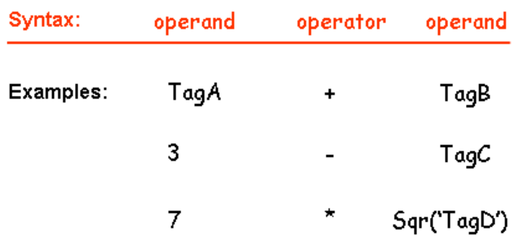
4.1 Performance Equations Syntax

Performance Equations syntax includes the following topics:

4.1.1 Performance Equation Syntax

Writing a Performance Equation calculation expression is very similar to writing an expression in arithmetic. In fact, you can use any of the standard arithmetic operators in a PE expression (such as **+, -, or ***) to add the values of two points together, add a number to the value of a point, etc.

As with arithmetic expressions, the building blocks of a PE calculation expression are operands and operators. Performance equations are simply expressions in which operators act on operands. A basic PE expression takes the form: `operand operator operand` – as shown here:

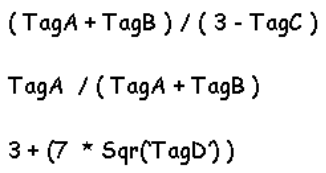


Syntax:	operand	operator	operand
Examples:	TagA	+	TagB
	3	-	TagC
	7	*	Sqr('TagD')

The PE Scheduler evaluates the first example as the value of *TagA plus the value of TagB*. The second example is *3 minus the value of TagC*. The third example is *7 times the square root of TagD.*

You can construct more complex PE expressions, just as you can in arithmetic. The PE Scheduler performs most operations in the same order as they'd be performed in a mathematical expression. For a complete listing of PE operator priority, see *Operator Priority*, on page 63.

Use parentheses to group together expressions you want to evaluate first:



```
( TagA + TagB ) / ( 3 - TagC )

TagA  / ( TagA + TagB )

3 + (7  * Sqr('TagD') )
```

The first example above evaluates as the sum of the values of *TagA and TagB, divided by the difference of 3 minus TagC*. The second example is *TagA divided by the sum of TagA and TagB.* The third example is *3 plus the product of 7 and the square root of TagD.*

4.1.2 Performance Equation Operands

The operands that the PE Scheduler recognizes are listed in Table 4–1. (As indicated under *Syntax Requirements*, certain operands must always be enclosed in single or double quotes.)

Table 4–1. Operands in Performance Equations

Operand Type	Syntax Requirements	Examples

Numbers	(none)	`1342`
		`98.6`
		`.0015`
		`1.2e2`

Tagnames	In single quotes	`'sinusoid'`
		`'ba:level.1'`
		`'ba.phase.1'`

PI Time Expressions	In single quotes	`'01-dec-03'`
		`'16-jul-94'`
		`'*'`

Strings	In double quotes	`"string string string"`
		`"Now is the Winter of Our discontent…"`
		`"sinusoid"`

Functions	Must be a PE function	`TagVal('sinusoid')`
		`TagAvg('sinusoid')`
		`Cos('sinusoid')`

Number Operands

You can use numbers in Performance Equations. The PE Scheduler processes all numbers as floating point numbers. Examples of numbers include:

```
3.14159
299792458
299792458.
0.671
.671
6.71e-1
```

Note: The second and third examples are equal, as are the fourth, fifth, and sixth.

4.1.3 Tagname Operands

You can use tagnames in Performance Equations to represent values from the Snapshot. You must put the tagname in single quotes, unless you are using the tagname as a string, in which case you must enclose it in double quotes. The PE Scheduler evaluates the tagname according to its use it in an expression, as a function argument, or as a time expression.

Tagnames in Expressions

If you use the tagname in an expression, PE evaluates the tagname as that point's value at the current time. For example:

```
3 + 'sinusoid'
```

is equal to the value of *sinusoid* at the current time (see note), plus three. The same value results from the expression:

```
3 + TagVal('sinusoid', '*')
```

Note: The exception is when this expression is used in a PE point, the PE point is event-based, and the **Location3** attribute is set to one.

Tagnames as Function Arguments

If you use the tagname as an argument in one of the PE built-in functions, then the PE subsystem evaluates the tagname according to the type of value expected by that particular function.

For example, if the function expects a tagname, then PE passes a tagname to the function. If the function expects any other data type such as a string or a number, PE Scheduler gets the current value of the point and passes that to the function—as whatever data type is expected.

For example, the **Concat** function expects two or more strings as arguments. It concatenates all the arguments into a single string:

```
Concat('sinusoid', 'ba:level.1')
```

To evaluate this expression, the PE Subsystem takes the current value of the *sinusoid* point and the *ba:level.1* point and passes these to the **Concat** function as strings. **Concat** then returns a string that is composed of the value of the *sinusoid* point followed by the value of the *ba:level.1* point. If the current values of these points are, respectively, 85.329 and 30.478, **Concat** returns the following string:

```
85.32930.478
```

Tagnames that Are Valid Time Expressions

Wherever possible, choose tagnames that cannot possibly be interpreted as time expressions. The tagname *t-151d*, for example, is also a valid time expression meaning *today minus 151 days*. If you must work with tagnames that are also valid time expressions, use the built-in function **TagNum** to ensure that the PE subsystem does not treat the tagname as a time. For example, `Abs(TagNum("t-151d"))` would return the absolute Snapshot value of point *t-151d*. Note that **TagNum** interprets a double-quoted string as the argument.

To the PE subsystem, an expression within single quotes can correspond either to a time or a tagname. The PE Scheduler treats expressions in single-quotes as tagnames for all the built-in functions that take a point as the first argument. (Examples include **TagVal, TagAvg,** and **AlmCondition**). In all other cases, the PE subsystem first attempts to resolve the expression as a time. If the expression is not a valid time, then the PE subsystem tries to resolve it to a tagname. If the point does not exist, the subsystem returns the error **Calc Failed**.

For example, `TagVal('t-151d')` returns the Snapshot value for the point *t-151d*, if it exists. However, the expression `t-151d` returns the date corresponding to 151 days before today—because it is a valid time expression.

4.1.4 String Operands

Strings are sequences of any printable characters. Strings are always enclosed in double-quotes. Some examples are:

```
"This is a string"
"sinusoid"
"14-Dec-97"
```

Note: Character strings might look like tagnames or time expressions, as in the second and third examples above. In some cases, the string in the third example might be interpreted as a time. The difference is that a character string is enclosed in double quotes (for example, *"string"*) while a tagname must be enclosed in single quotes (for example, *'tagname'*) and a time expression may be enclosed in either single or double quotes.

4.1.5 Time Expression Operands

In a PE, you can use any standard PI time expression if you enclose it in single quotes. The following topics briefly explain PI time expressions and how to work with them in PEs. For more information about PI time, see the ***PI Server System Management Guide***.

- *PI Time Expressions*
- *Tips for Working with PI Time Expressions*
- *Times as Strings*
- *Quick Reference Table of Time Syntax Examples*

PI Time Expressions

PI allows three basic types of time expressions: *relative time*, *combined time*, and *absolute time*.

Relative Time

Relative time expressions are some number of a number of days, hours, minutes, or seconds, specified with either a leading plus sign or a leading minus sign.

The reference time, or starting time, for the relative time expression is usually the current time. In PEs, we recommend you use a combined time expression, rather than a relative time expression.

```
+1d
-24h
-3m
+24s
```

Combined Time

A combined time expression is a specific reference time, followed by a relative time expression. In Performance Equations, you enclose the combined time expression in single quotes (or double quotes, if you are passing the time expression to a PE function as a string).

```
'*+8h'
'18-dec-02 -3m'
'T+32s'
```

Combined time expressions should contain only a single operator. If you add additional operators, you get unpredictable results. For example, the following are not valid time expressions:

```
'*+1d+4h'
'T-1d+12h'
```

Absolute Time

Absolute time expressions are everything else—in other words, any time expression that is neither a relative nor a combined time expression is an absolute time expression. When using absolute time expressions in PEs, put the time expression in single quotes.

```
'*'
'14-Dec-97'
TagVal('Sinusoid', "1-Jun-2000")
'11-Nov-96 2:00:00.0001'
't'
```

Tips for Working with PI Time Expressions

When working with time expressions in PEs, please follow these important guidelines:

- Use absolute or combined time expressions, rather than relative time expressions. If you don't, depending on the context of the expression, you might get an error message or PI might choose a starting time that is not what you expect.
- Relative and combined time expressions do not provide any special processing for clock or calendar events such as daylight savings time boundaries. If you need intervals based on local clock time, use **Noon( )** and **Bod( )** functions.
- Relative and combined time expressions contain only a single operator: either a single plus sign **(+)** or a single minus sign **(-).** If you add additional operators, you get unpredictable results. For example, the following are not valid time expressions:

  ```
  '*+1d+4h'
  'T-1d+12h'
  ```

Times as Strings

You can also pass a time expression as a string to a function that expects a string. In this case, enclose the time expression in double quotes, rather than single quotes.

Performance Equations Time Syntax Reference

The following table provides PI time syntax examples.

Table 4–2. Examples of Time Syntax

Description	Code	Example	Example Description
Now	`*`	`*`	Now—current time
Midnight at specified date and current month	`dd`	`25`	Midnight the 25th of the current month
Midnight at specified date	`dd-mmm-yy`	`25-aug-02`	Midnight on August 25th 2002
Specified time at specified date	`dd-mmm-yy hh:mm:ss`	`25-aug-02 12:00:00`	Noon on August 25th 2002
Specified hour at current date	`h:`	`8:`	8:00 at current date
Specified hour and day of current month, year and minute	`dd h:`	`25 8:`	8:00 on the 25th day of the current month
Today at 00:00:00	`t`	`t+7h`	Today at 7 a.m.
Yesterday at 00:00:00	`y`	`y+15h`	Yesterday at 3 p.m.
Day of the week at 00:00:00	`sa, su, mo, tu, w, th, f`	`mo+6.5h`	Monday at 6:30 a.m.
Time interval (days)	`#.#d`	`1.3d`	1.3 days
Time interval (hours)	`#.#h`	`1.5h`	One hour and a half
Time interval (minutes	`#.#m`	`32m`	32 minutes
Time interval (seconds)	`#.#s`	`49s`	49 seconds

4.1.6 Function Operands

The PE Scheduler provides built-in functions that perform a variety of operations. You can use any of these functions as an operand in a Performance Equation.

Numbers and Strings as Digital States

Digital state values consist of a state set specifier and a state number within that set. Each set has a list of text names for the states. You can set a digital point with an expression that results in either a number (specifying the offset) or a string (specifying the state name).

Comparing the Value of Digital and Numeric Points to Strings

In PEs, you can use expressions that compare the value of a digital or numeric point to a string. For example, if the string "*Run*" is in the state set for digital point *PumpStatTag*, then the following expression is valid:

```
If 'PumpStatTag' <> "Run" then 1 else 0
```

The state set for a numeric point is the System State Set. The System State Set is the default state set for all points and it contains a collection of all the states that any point can use. Examples are *Shutdown*, *Over Range*, *I/O Timeout*, etc. For example, the expression

```
'sinusoid' = "Shutdown"
```

is true if the numeric point *sinusoid* contains the digital state **Shutdown** from the System Digital State Set.

Comparing a Digital State to a String Point

If you want to compare a digital state to a string point, use the **DigState** function to convert a string to a digital state explicitly. For example, the following are different:

```
If 'StringTag' = "Shutdown" then 0 else 1
If 'StringTag' = DigState("Shutdown") then 0 else 1
```

The former is true if the string point contains the string "*Shutdown*" while the latter is true if the point contains the digital state **Shutdown**.

Setting the Digital State for a Numeric or Digital Point

You can use a string to set the digital state for a numeric or digital point. When you do this, PE Scheduler looks first in that point's state set for a state that corresponds to the string. If the state does not exist in the point's state set, PE Scheduler searches the System Digital State Set for the state string. If PE Scheduler cannot find the state string in either the Digital State Set for that point or in the System Digital State Set, it returns **Calc Failed**. The state set for a numeric point is the System Digital State Set.

4.1.7 List of all Performance Equation Operators

You use PE operators in PE expressions to act on operands such as tagnames, numbers, and time expressions. Operator priority works basically as it does in math—multiplication and division are performed before addition and subtraction, etc. You can also use parentheses to group operations, just as you do in math. For a complete explanation of operator priority, see *Operator Priority* on page 63.

Table 4–3 lists all the PE operators, according to type, with examples.

Table 4–3. PE Operators, Listed by Type, with Examples

Operator Type	Operator	Syntax Example	Meaning (A, B, C and D are all operands)
Arithmetic	+	`A + B`	Addition: A + B

Operator Type	**Operator**	**Syntax Example**	**Meaning (A, B, C and D are all operands)**
	`-`	`A - B`	Subtraction: A minus B
	`*`	`A * B`	Multiplication: A times B
	`/`	`A / B`	Division: A divided by B
	`^`	`A ^ B`	Raising to a power: A to the power of B (AB)
	`Mod`	`A mod B`	Modulus: the remainder of A divided by B
Relational	`<`	`A < B`	Less than: returns true if A is less than B
	`=`	`A = B`	Equal to: returns true if A equal to B
	`>`	`A > B`	Greater than: returns true if A is greater than B
	`<=`	`A <= B`	Less than or equal to: returns true if A is less than or equal to B
	`<>`	`A <> B`	Not equal to: returns true if A is not equal to B
	`>=`	`A >= B`	Greater than or equal to: returns true if A is greater than or equal to B
Prefix	`Not`	`NOT A`	Complementation: returns true if A is 0 and False otherwise
	`-`	`- A`	Negation (as prefix operator): returns the negative of A
Conjunction, Disjunction and Inclusion	`And`	`A and B`	Conjunction: returns true if operands A & B both evaluate to true. If both A and B are integers, returns the result of a bitwise AND operation.
	`Or`	`A or B`	Inclusive disjunction: returns true if either operand A or operand B evaluates to true. If both A and B are integers, returns the result of a bitwise OR operation.
	`in ..`	`A in B..D`	Membership in a range: returns true if the value of A is between B and D
	`in ( )`	`A in (B1, B2, …BN)`	Membership in a discrete set: returns true if the value of A matches any of the values enclosed in the parentheses.
If-Then-Else Expressions	`if then else`	`if A then B else D`	If-then-else expression: returns B if A is true—otherwise it returns D

Arithmetic Operators

PE operators include the simple arithmetic operators in Table 4–4.

Table 4–4. PE Arithmetic Operators

Operator	Meaning	Notes
+	Addition	
–	Subtraction	
*	Multiplication	
/	Division	
^	Raising to a power	
mod	Modulus	The mod operator returns the remainder after its left operand is divided by its right operand. For example, 17 mod 3 equals 2.

For a complete list of all PE operators, see *List of all Performance Equation Operators* on page 57.

Arithmetic Operations on Time Values

You can perform certain arithmetic operations on times, such as adding two time expressions, or subtracting one absolute time expression from another. The result of the operation is one of the following:

- **A Timestamp**. A timestamp is just a date and time in the PI timestamp format. For example: `25-aug-02 12:00:00`
- **A Period.** A period is a number of seconds.
- **A Number**.

Table 4–5 shows valid operations and results, where **N** represents a number, **T** represents an absolute or combined time expression, and **P** represents a period.

Table 4–5. Valid Operations on Time Values

Operator	Expression	Result	Example
+	`T+ P`	`T`	`'*' + '+3h'`
	`T+ N`	`T`	`'*' + 10`
	`P + N`	`P`	`('t'- 'y') + 10`
	`P + P`	`P`	`('t'- 'y') + ('t'-'y')`
- (infix)	`T - P`	`T`	`'*' - '+3h'`

Operator	Expression	Result	Example
	`T - N`	`T`	`'*' - 10`
	`T - T`	`P`	`'t' - 'y'`
	`P - N`	`P`	`('t'-'y') - 10`
	`P - P`	`P`	`('t'-'*') - ('t'-'y')`
*****	`P * N`	`P`	`('t' -'y') * 5`
	`N * P`	`P`	`5 * ('+1d' -'+1h')`
/	`P / P`	`N`	`('t'-'*') / ('t'-'y')`
	`P / N`	`P`	`('t'-'*') / 2`
	`N / P`	`N`	`2 / ('t'-'*')`
mod	`T mod P`	`T` (see note)	`'*' mod ('*'-'t')`
	`T mod N`	`T` (see note)	`'*' mod 2`
	`P mod P`	`P`	`('*'-'y') mod ('*'-'t')`
	`P mod N`	`P`	`('*'-'y') mod 3`
- (prefix)	`-P`	`P`	`-('*'-'y')`

Note: The timestamp returned is the result of T mod P or T mod N added to January 1, 1970 Universal Coordinate Time (UTC). So depending on the time zone, different results are expected; in some case, even an error value is returned. In PI for OpenVMS systems, the use of *T mod P* or *T mod N* returns *P*.

Relational Operators

A relational operator (one of **<, =, >, <=, <>,** and **>=**) returns a value of **0** for false or **1** for true. You can use these operators to compare numbers, times, digital states, or character strings. With relational operators, you can compare bad values, or values of different types without generating an error.

Table 4–6. Relational Operators in Performance Equations

Operator	Meaning
<	Less than
=	Equal to
>	Greater than
<=	Less than or equal to
<>	Not equal to
>=	Greater than or equal to

Comparing Bad Values

If you're comparing two operands of the same type, and one or both operands has a bad value, the expression returns **0**, rather than an error value.

Comparing Operands of Different Types

When you use the <> operator to compare any two operands of different types, the expression always returns a **1** (i.e., 'true'). When you use any other relational operator (anything except <>) to compare two operands of different types, the expression returns a **0** (i.e., 'false') except in the following cases:

- If one of the two operands is the digital type, then the PE subsystem compares the digital operand to the digital state of the other operand, if it exists. For example:

  ```
  'sinusoid' = DigState("Shutdown")
  ```

- If the *sinusoid* point has a digital state **Shutdown**, then this expression returns a value of **1** (i.e., true). Otherwise, it returns a value of **0** (i.e., false)
- If one of the two operands is the string type and the other is neither digital nor string type, then the PE subsystem compares the string operand to the digital state of the other operand, if it exists. This allows the string substitution of its corresponding digital state; i.e., **Shutdown** and **DigState**("**Shutdown**") would be the same.

Time Comparisons

You can perform all comparisons, including **in**, on times.

```
'*+20m' >= '*+300s'
PrevEvent('tag1', '*') > PrevEvent('tag2', '*')
```

If a comparison is true, the result is **1**; otherwise, it is **0**.

Prefix Operators

A prefix operator is simply an operator that appears to the left of its operand, for example, `"-x"`.

Table 4–7. Prefix Operators

Operator	Meaning
–	Negation
Not	Complementation: Returns 1 if operand is 0 (or rounding to 0) and 0 otherwise

The expression following a prefix operator should be numeric. If you use a tagname as the operand, make sure that the point returns a number. Note too, that even points that typically return a number, sometimes return a digital state, such as **Shutdown**. Valid examples include:

```
-3
Not 7
-TagVal ('sinusoid')
Not Cos('ba:level.1')
```

```
-StateNo('DigitalTag')
Not TagBad('StringTag'))
```

The last two examples use digital and string points (*DigitalTag* and *StringTag*) but these are used as arguments to functions that return numbers (**StateNo** and **TagBad**).

Conjunction, Disjunction and Inclusion Operators

You can use **and, or, in ..**, and **in()** operators in PEs.

Table 4–8. Conjunction, Disjunction and Inclusion Operators

Operator	Meaning	Returns
and	Conjunction	Returns true if operands A & B both evaluate to true. If both A and B are integers, returns the result of a bitwise AND operation.
or	Inclusive disjunction	Returns true if either operand A or operand B evaluates to true. If both A and B are integers, returns the result of a bitwise OR operation.
in ..	Membership in a range	The in .. operator returns 1 if true and 0 if false.
in ( )	Membership in a discrete set	The in () operator returns 1 if true and 0 if false.

Inclusion Operator Examples

The following are two examples that use inclusion operators.

```
If 1 in 0 .. 2 Then 1 Else 0
```

The result is **1**, since **1** is between **0** and **2**.

```
If 1 in (0, 2) Then 1 Else 0
```

The result is **0**, since **1** does not equal either **0** or **2**.

Using the Inclusion Operator with Digital State Functions

You can use the **in ..** operator with functions that return digital states, in which case the operator uses the offset within the Digital State Set for comparison. The digital states must all be in the same Digital State Set. Lexical comparisons are made with character strings.

Time Comparisons

You can use the **Inclusion operators (in.., in())** on time expressions. If the comparison is true, the result is **1**; otherwise, it is **0**.

If-Then-Else Expressions

The **if–then–else** operator is a compound operator whose operands are used as follows:

```
if expr0 then expr1 else expr2
```

where **expr0**, **expr1**, and **expr2** are expressions. If **expr0** is true (nonzero), this operator returns the value of **expr1**; it returns the value of **expr2** if **expr0** is false (zero).

Here are some examples:

```
if 'tag1' > 50 then "overlimit" else "good"
if 'tag1'= 1 then Sin('tag2') else if 'tag1'= 2 then Cos('tag2') else
Tan('tag2')
if 'tag3'<> "shutdown" then (if 'tag1' > 'tag2' then 'tag1' else 'tag2') else
"error"
'*' + (if 'tag2' = 0 then 3600 else 0)
```

Note: You must include the **'if,'** the **'then,'** and the **'else.'** Nested operations are supported.

4.1.8 Operator Priority

The PE Scheduler executes operators in order of priority, from highest to lowest. When multiple operators have the same priority, the order of execution is either from left-to-right or right-to-left, depending on the operator, as listed in the following table.

Table 4–9. Operator Priority

Operator	Priority	Order
-(prefix)	9 (done first)	L-R
^	8	R-L
Not	7	L-R
*, /, mod	6	L-R
+ , -	5	L-R
<, =, >, <=, <>, >=	4	L-R
in ..., in( )	3	L-R
And	2	L-R
Or	1 (done last)	L-R

Note: The **Not** operator has a priority between **^** and *****. This differs from conventional priority schemes.

You can use parentheses anywhere to affect the order of calculation. Regardless of operator priority, operations within parentheses are evaluated before operations outside those parentheses. For example, `(2+3) * 5 equals 25` while `2 + 3 * 5 equals 17`.

4.1.9 Data Types

The PE Scheduler recognizes the following data types:

- Number
- Character string
- Tagname
- Time
- Period

Every *variable* has one of these data types; every *function* returns one of these data types. The PE Scheduler cannot typically use one data type where another is expected. For example, you cannot add two character strings, or multiply two times together. Additionally, the built-in functions might require particular data types for particular arguments.

Type Checking

At compile time, the PE subsystem checks type compatibility as far as possible. This process catches some errors, such as trying to add a number to a character string.

However, not all type compatibility errors can be detected at compile time. The expression

```
'sinusoid' / 2.0
```

works well if *sinusoid* has a numeric value, but if *sinusoid* is equal to the digital state **Shutdown** the expression returns an error (**Calc Failed**).

Note: Comparisons (expressions using relational operators) are an exception to this rule. Every comparison is valid, regardless of its operand types.

4.1.10 Error Values

When the PE subsystem cannot perform a calculation during runtime, it returns the error value **Calc Failed**. Error values propagate through most calculations. For example, an error value plus one is an error value. Exceptions to this rule are:

- The **BadVal** and **Concat** functions
- The relational operators when a value of **0** is returned

To check for compile time errors on Windows-based computers, check the *pipc.log* file located in the *\pipc\dat\* directory. For UNIX check the *pipeschd.log* file located in *C:\Program Files\Rockwell Software\FactoryTalk Historian\Server\PI\log*.

4.1.11 Test the Performance Equation Syntax

The **pipetest** utility is a command line function that checks the syntax of a Performance Equation. There is also an System Management Tool (SMT) plug-in to test a performance equation. (See the SMT documentation for information.) It can operate interactively, take its input from a file or check the extended descriptor of a point.

Run the pipetest Utility

The **pipetest** utility is located in the *pi\adm* directory. To start **pipetest**, open a command window, change to the *pi\adm* directory, and type a **pipetest** command. For a complete list of **pipetest** commands, type:

```
pipetest -?
```

The **pipetest** utility is limited to equations that are 4095 characters or less and you can not use it to test dynamic response functions.

Using pipetest in Interactive Mode

To run the **pipetest** utility interactively from a command prompt window, open a command window, change to the *pi\adm* directory, and type:

```
pipetest
```

At the **pipetest** equation prompt, type in the equation you want to test. If the equation syntax is not valid, **pipetest** displays a syntax error. If the syntax is valid, **pipetest** displays the result of the equation.

Using pipetest in File Input Mode

You can also put one or more performance equations in a simple text file, and pass the entire file to **pipetest** using the **–f** switch. In the text file, you put each equation on a single line. You can include comment lines by beginning the line with an exclamation mark (**!**).

Here's the text from an example **pipetest** file, called *peTestEquations.txt*:

```
! test calculation for point A
if BadVal('sinusoid') then 0 else ('sinusoid')/25
! test calculation for point B
TimeLT('sinusoid', 'y' , 't', TagVal('sinusoid', '*'))
```

To test the equations in the file, type:

```
pipetest -f peTestEquations.txt
```

Each input line in turn is echoed and the evaluated result is displayed.

Check a Point's ExDesc Parameter

To check the Performance Equation of a specific point or group of points, use the **–t** switch followed by a tag mask. For example:

```
pipetest -t sinusoi*
```

will process all points whose tagnames begin with the letters *"sinusoi"*. The **pipetest** utility echoes the tagname, the value of the extended descriptor field, and the evaluated result. The **pipetest** utility ignores the **event=tagname** expression so, for example, the expression `event='sinusoid', 1+2` is the same as `1+2`.

Optionally, you can also specify a **PointSource** character. For example, to process all points with a point source character of **C**, type:

```
pipetest -f * C
```

4.2 Performance Equations Functions

In addition to all the basic arithmetic operators, the PE subsystem provides a large number of built-in functions that you can use to perform more complex operations, such as taking the sine or cosine of a point value, taking the average of a tag's value over time, etc.

4.2.1 Function Arguments

Functions have one or more *arguments*, or *inputs*, which are enclosed in parenthesis following the function name. Some of the arguments may be optional. If there are several arguments, they are separated by commas:

```
functionName(argument1, argument2, argument3)
```

The following are examples of function expressions:

```
Max(3, 5, 12.6, 'sinusoid')
PrevEvent('sy:arc001', '*-2h')
Sqr(Abs(TagMax('tag', 'y', 't')))
Log(if 'tag'=2 then .5 else .2)
```

Functions can also be nested and joined in expressions:

```
Avg(TagVal('TagA', 'y'), TagVal('TagB', 'y'), TagVal('TagC', 'y') )
if TagVal('TagA', '*') < TagVal('TagB', '*') then sin('TagB') else sin('TagA')
```

You can use a tagname in any argument where a number or character string is called for. A tagname in single quotes is evaluated as if it had been written **TagVal**(tagname), which is the same as **TagVal**('tagname', '*'). This gets the point's value at the "current" time for the calculation.

If the argument calls for a number, but the point's value is a digital state when the function is evaluated, a run-time error (**Calc Failed**) is generated.

4.3 List of Built-in Functions

The PE Scheduler provides a wide range of built-in functions that make it easier for you to perform calculations on PI data. (Note that you can also use Steam Table Functions in PEs.)

Note: Immediately below, are two tables that provide a complete listing of all built-in PE functions. One table lists functions grouped by type, and the the other is arranged alphbetically.

4.3.1 Functions Grouped By Type

Table 4–10 lists all functions categorically, as follows.

- Math functions

- Other Math Functions
- Aggregate Functions
- Miscellaneous Functions
- PI Archive Retrieval
- PI Archive Search
- PI Archive Statistics
- Point Attributes
- Time Functions
- Dynamic Response
- Alarm Status Functions
- String Functions
- String Conversion

Table 4–10. Functions Grouped by Type

Function Type	Name	Meaning
Math Functions	Asin	Arc sine
	Acos	Arc cosine
	Atn	Arc tangent
	Atn2	Arc tangent (two arguments)
	Cos	Cosine
	Cosh	Hyperbolic cosine
	Exp	Exponential
	Log	Natural logarithm
	Log10	Common logarithm
	Sin	Sine
	Sinh	Hyperbolic sine
	Sqr	Square root
	Tanh	Hyperbolic tangent
	Tan	Tangent
Other Math Functions	Abs	Absolute value
	Float	Conversion of string to number
	Frac	Fractional part of number
	Int	Integer part of number
	Poly	Evaluate polynomial
	Round	Round to nearest unit
	Sgn	Numerical sign

Function Type	Name	Meaning
	Trunc	Truncate to next smaller unit
Aggregate Functions	Avg	Average
	Max	Maximum
	Median	Median selector
	Min	Minimum
	PStDev	Population standard deviation
	SStDev	Sample standard deviation
	Total	Sum
Miscellaneous Functions	BadVal	See if a value is bad (not a number or time)
	Curve	Get value of a curve
	DigState	Get digital state from a string
	IsDST	Test whether a time is in local daylight savings time period
	IsSet	Test if a PI value is annotated, substituted, or questionable
	StateNo	The code number of a digital state
	TagBad	See if a point has an abnormal state
PI Archive Retrieval	NextEvent	Time of a point's next Archive event
	NextVal	Point's next value after a time
	PrevEvent	Time of a point's previous Archive event
	PrevVal	Point's previous value before a time
	TagVal	Point's value at a time
PI Archive Search	FindEq	Timestamp when point = value
	FindGE	Timestamp when point >= value
	FindGT	Timestamp when point > value
	FindLE	Timestamp when point <= value
	FindLT	Timestamp when point < value
	FindNE	Timestamp when point ~= value
	TimeEq	Total period when point = value
	TimeGE	Total period when point >= value
	TimeGT	Total period when point > value
	TimeLE	Total period when point <= value
	TimeLT	Total period when point < value
	TimeNE	Total period when point ~= value
PI Archive	EventCount	Number of Archive events

Function Type	**Name**	**Meaning**
Statistics	PctGood	Percent of good time in a period
	Range	Range of minimum to maximum value
	StDev	Time-weighted standard deviation
	TagAvg	Time-weighted average
	TagMean	Event-weighted average
	TagMax	Maximum value in a period
	TagMin	Minimum value in a period
	TagTot	Time integral over a period
Point Attributes	TagDesc	Get a point's descriptor
	TagEU	Get a point's engineering unit string
	TagExDesc	Get a point's extended descriptor
	TagName	Get a point's name
	TagNum	Get a point's ID
	TagSource	Get a point's point source character
	TagSpan	Get a point's span
	TagType	Get a point's type character
	TagTypVal	Get a point's typical value
	TagZero	Get a point's zero value
Time Functions	Bod	Timestamp for beginning of the day for given time
	Bom	Timestamp for beginning of the month for given time
	Bonm	Timestamp for first of the next month for given time
	Day	Day of the month from a time
	DaySec	Seconds since midnight from time
	Hour	Hour from a time
	Minute	Minute from a times
	Month	Month from a time
	Noon	Timestamp for local noon of day of a times
	ParseTime	Convert character string to time
	Second	Second from a times
	Weekday	Day of the week from a times
	Year	Year from a time
	Yearday	Day of the year from a time

Function Type	**Name**	**Meaning**
Dynamic Response	Arma	Dynamic response from Auto Regressive Moving Average model
	Delay	Introduce time delay
	MedianFilt	Select the median value of time series
	Impulse	Dynamic response characterized by impulse response shape
Alarm Status Functions	AlmAckStat	Alarm acknowledgement status code
	AlmCondition	Condition code number for Alarm State
	AlmCondText	Alarm condition as text
	AlmPriority	Alarm priority number
String Functions	Ascii	ASCII character code for a character
	Char	String for ASCII character code(s)
	Compare	Wild comparison of two strings
	DigText	Text for a digital state
	Format	Formatting of a numerical number
	InStr	Instance of a sub-string
	LCase	Conversion of all characters to lower case
	Len	Length of a string
	Left	First characters in a string
	LTrim	Removal of blanks on the left side of a string
	Mid	Extraction of a sub-string from a string
	Right	Last characters in a string
	RTrim	Removal of blanks on the right side of a string
	Trim	Removal of blanks on both sides of a string
	UCase	Conversion of all characters to upper case
String Conversion	Concat	Concatenate two or more strings
	String	String representing any PI value
	Text	Concatenation of strings for a series of PI value arguments

4.3.2 Functions Listed Alphabetically

Table 4–11. Functions Listed Alphabetically

Name	**Meaning**
Abs	Absolute value

Name	**Meaning**
Acos	Arc cosine
AlmAckStat	Alarm acknowledgement status code
AlmCondition	Condition code number for Alarm State
AlmCondText	Alarm condition as text
AlmPriority	Alarm priority number
Arma	Dynamic response from Auto Regressive Moving Average model
Ascii	ASCII character code for a character
Asin	Arc sine
Atn	Arc tangent
Atn2	Arc tangent (two arguments)
Avg	Average
BadVal	See if a value is bad (not a number or time)
Bod	Timestamp for beginning of the day for given time
Bom	Timestamp for beginning of the month for given time
Bonm	Timestamp for first of the next month for given time
Char	String for ASCII character code(s)
Compare	Wild comparison of two strings
Concat	Concatenate two or more strings
Cos	Cosine
Cosh	Hyperbolic cosine
Curve	Get value of a curve
Day	Day of the month from a time
DaySec	Seconds since midnight from time
Delay	Introduce time delay
DigState	Get digital state from a string
DigText	Text for a digital state
EventCount	Number of Archive events
Exp	Exponential
FindEq	Timestamp when point = value
FindGE	Timestamp when point >= value
FindGT	Timestamp when point > value
FindLE	Timestamp when point <= value
FindLT	Timestamp when point < value

Name	**Meaning**
FindNE	Timestamp when point ~= value
Float	Conversion of string to number
Format	Formatting of a numerical number
Frac	Fractional part of number
Hour	Hour from a time
Impulse	Dynamic response characterized by impulse response shape
InStr	Instance of a sub-string
Int	Integer part of number
IsDST	Test whether a time is in local daylight savings time
IsSet	Test if a PI value is annotated, substituted, or questionable
LCase	Conversion of all characters to lower case
Len	Length of a string
Left	First characters in a string
Log	Natural logarithm
Log10	Common logarithm
LTrim	Removal of blanks on the left side of a string
Max	Maximum
Median	Median selector
MedianFilt	Select the median value of time series
Mid	Extraction of a sub-string from a string
Min	Minimum
Minute	Minute from a time
Month	Month from a time
NextEvent	Time of a point's next Archive event
NextVal	Point's next value after a time
Noon	Timestamp for local noon of day of a time
ParseTime	Convert character string to time
PctGood	Percent of good time in a period
Poly	Evaluate polynomial
PrevEvent	Time of a point's previous Archive event
PrevVal	Point's previous value before a time
PStDev	Population standard deviation
Range	Range of minimum to maximum value

Name	Meaning
Right	Last characters in a string
Round	Round to nearest unit
RTrim	Removal of blanks on the right side of a string
Second	Second from a time
Sgn	Numerical sign
Sin	Sine
Sinh	Hyperbolic sine
Sqr	Square root
SStDev	Sample standard deviation
StateNo	The code number of a digital state
StDev	Time-weighted standard deviation
String	String representing any PI value
TagAvg	Time-weighted average
TagBad	See if a point has an abnormal state
TagDesc	Get a point's descriptor
TagEU	Get a point's engineering unit string
TagExDesc	Get a point's extended descriptor
TagMax	Maximum value in a period
TagMean	Event-weighted average
TagMin	Minimum value in a period
TagName	Get a point's name
TagNum	Get a point's ID
TagSource	Get a point's point source character
TagSpan	Get a point's span
TagTot	Time integral over a period
TagType	Get a point's type character
TagTypVal	Get a point's typical value
TagVal	Point's value at a time
TagZero	Get a point's zero value
Tan	Tangent
Tanh	Hyperbolic tangent
Text	Concatenation of strings for a series of PI value arguments
TimeEq	Total period when point = value

Name	**Meaning**
TimeGE	Total period when point >= value
TimeGT	Total period when point > value
TimeLE	Total period when point <= value
TimeLT	Total period when point < value
TimeNE	Total period when point ~= value
Total	Sum
Trim	Removal of blanks on both sides of a string
Trunc	Truncate to next smaller unit
UCase	Conversion of all characters to upper case
Weekday	Day of the week from a time
Year	Year from a time
Yearday	Day of the year from a time

4.4 Performance Equations Functions Reference

Abs

Return the absolute value of an integer or real number.

Format

```
Abs(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The absolute value of **x**.

Exceptions

If **x** is not an integer or real number, returns an error value.

Examples

```
Abs(1)
Abs(-2.2)
Abs('tag1')
Abs('tag1'- 'tag2')
```

Acos

Return the inverse (or arc) cosine of an integer or real number. The inverse cosine of **x** is the angle in radians whose cosine is equal to **x**.

Format

```
Acos(x)
```

Arguments

```
x
```

Must be a real number between -1.0 and 1.0, inclusive.

Returns

The inverse cosine of **x**, in radians.

Exceptions

If **x** is not a number, or is less than **-1.0** or greater than **1.0**, returns an error value.

Examples

```
Acos(1-.5)
Acos(-.5)
If 'tag1' < 1 and 'tag1' > -1 then Acos('tag1') else 0
```

See Also

Asin, Atn, Atn2, Cos, Cosh, Sin, Sinh, Tan, Tanh

AlmAckStat

Return the acknowledgement status code for an alarm point.

Format

```
AlmAckStat(alm)
```

Arguments

```
alm
```

An alarm tagname.

Returns

The acknowledgement status code for the given Alarm State. Possible values are:

0 - acknowledged (or no alarm)

1- unacknowledged

2 - missed alarm

Exceptions

If the argument is not a digital state tagname, an error condition is returned.

Examples

```
AlmAckStat('alarmtag')
```

See Also

AlmCondition, AlmCondText, AlmPriority

AlmCondition

Returns the condition code for an Alarm State.

Format

```
AlmCondition(alm)
```

Arguments

```
alm
```

An alarm tagname.

Returns

The code number of the condition for the alarm tagname.

Exceptions

If the argument is not a digital state tagname, an error is returned.

Examples

```
AlmCondition('alarmtag')
```

See Also

AlmAckStat, AlmCondText, AlmPriority

AlmCondText

Return the string for the condition of an Alarm State.

Format

```
AlmCondText(alm)
```

Arguments

```
alm
```

An alarm tagname.

Returns

The condition text for the condition represented by the alarm status.

Exceptions

If the argument is not a digital state tagname, an error condition is returned.

Examples

```
AlmCondText('alarmtag')
```

See Also

AlmAckStat, AlmCondition, AlmPriority

AlmPriority

Return the priority of an Alarm State.

Format

```
AlmPriority(alm)
```

Arguments

```
alm
```

A digital state value from a PI Alarm State Set.

Returns

The priority of the given alarm. Priorities are small positive integers. Priority **0** is an alarm that is never unacknowledged.

Exceptions

If the argument is not from a valid Digital State Set for alarms, an error condition is returned.

Examples

```
AlmPriority('alarmtag')
```

See Also

AlmAckStat, AlmCondition, AlmCondText

Arma

Calculate dynamic response for Auto Regressive Moving average model as shown in Figure 4–1.

Format

```
Arma(in, runflag, (a1, a2, … aN),(b0, b1, b2, … bN))
```

Arguments

The denominator and numerator coefficient series are enclosed in parenthesis with a comma between the two. There must be only one more term (**b0**) in the numerator than denominator. Both **in** and **runflag** may be any expression that evaluates to a number.

```
in
```

Must be an integer or real number.

```
runflag
```

Non-zero enables filter to run.

```
a1,a2,…
```

Coefficients of past output terms.

```
b0, b1,b2…
```

Coefficients of the present and past input terms of the model.

Returns

The next output value in time series response to past and present input.

```
ut = a1 * ut-1 + a2 * ut-2 + … + an * ut-n + b0 * yt +
b1 * yt-1 + b2 * yt-2 + … + bn * yt-n
```

Where **ut** is model output at time **t**, now. **ut-1** is output one sample interval in the past. **yt** is the input signal at time **t**. If **runFlag** expression result is **0**, the model is reset. Depending on the number of coefficients used, **Arma** stores the inputs and outputs until an evaluation of the model is available. For example, in

```
Arma( 'input_tag', 1, ( 0. ,1), ( 1, -1 ,1))
```

Arma stores two previous values of the input and output. Hence when the point is first created, no good results will be given until two prior values of the input have been stored. From then on, the two most current previous values will be stored.

Exceptions

Arma will give different results depending on which type of scheduling is used. In scan class scheduling, the interval between time series values depends on the scan class and gives values at evenly spaced time intervals. On the other hand, event based scheduling is dependent on a trigger from another point. If the exception deviation is not zero, the intervals for events will be not be evenly spaced in time and hence **Arma** will give results that are not trustworthy. **Arma** is not supported in the **pipetest** utility or in FactoryTalk Historian DataLink. If the input point is not a real number or integer, **Arma** will return an error.

Examples

```
Derivative: Arma('input_tag', 1, (0.), (1, -1))
Integration: Arma('input_tag', 1, (1.), (.05, 0.))
Second order filter: Arma('input_tag', 1, (.25,.25), (.1,.25,.15 ))
```

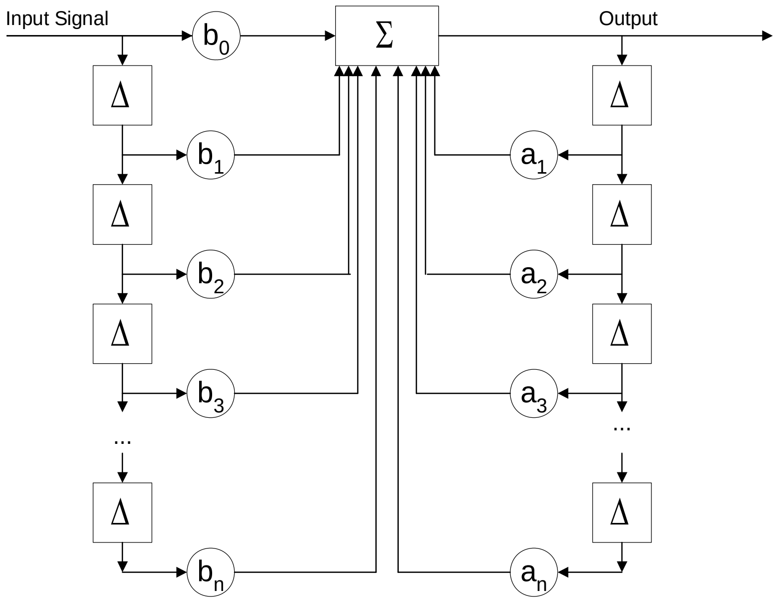


Figure 4–1. Block Diagram of ARMA Calculation Function

Ascii

Return the ASCII character code of the first character of a string.

Format

```
Ascii(String)
```

Arguments

```
string
```

Any expression evaluating to a string.

Returns

The character code of the first character of the string.

Exceptions

If the argument is not a string, an error value is returned.

Examples

```
Ascii( "D" ) = 68
Ascii( string('cdm158' ) )
```

Asin

Return the inverse (or arc) sine of a number. The inverse sine of **x** is the angle in radians whose sine is equal to **x**.

Format

```
Asin(x)
```

Arguments

```
x
```

Must be a real number between **-1.0** and **1.0**, inclusive.

Returns

The inverse sine of **x**, in radians.

Exceptions

If **x** is not a number, or is less than **-1.0** or greater than **1.0,** returns an error value.

Examples

```
Asin(TagVal('tag1','y'))
Asin(-0.5)
Asin('tag1')
```

See Also

Acos, Atn, Atn2, Cos, Cosh, Sin, Sinh, Tan, Tanh

Atn

Return the inverse (or arc) tangent of an integer or real number. The inverse tangent of **x** is the angle in radians whose tangent is equal to **x**.

Format

```
Atn(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The inverse tangent of **x**, in radians.

Exceptions

If x is not an integer or real number, returns an error value.

Examples

```
Atn(1)
Atn(-2.2)
Atn('tag1')
Atn('tag1'- 'tag2')
```

See Also

Acos, Asin, Atn2, Cos, Cosh, Sin, Sinh, Tan, Tanh

Atn2

Calculate the inverse (or arc) tangent of a pair of numbers, which represent a point on the plane. If you draw a line between the point whose Cartesian coordinates are (*a*, *b*) and the origin, the angle between that line and the *x*-axis is the inverse tangent of (*a*, *b*).

Format

```
Atn2(a, b)
```

Arguments

```
a
```

Must be an integer or real number.

```
b
```

Must be an integer or real number.

Returns

The inverse tangent of (*a*, *b*), in radians.

Exceptions

If *a* or *b* is not an integer or real number, returns an error value.

Examples

```
Atn2(TagVal('tag1', 'y'),TagVal('tag1', 'y'))
Atn2(1,1)
Atn2('tag1', 'tag2')
```

See Also

Acos, Asin, Atn, Cos, Cosh, Sin, Sinh, Tan, Tanh

Avg

Return the average of all the arguments.

Format

```
Avg(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be numbers, times, or periods but all must be the same data type.

Returns

The average of the arguments. The result is the same data type as the operands.

Exceptions

Arguments whose run-time values are character strings or digital states are not included in the average. If all values are character strings or digital states, *Avg* returns an error value.

Examples

```
Avg(TagVal('tag1','y'),TagVal('tag1', 'y'),1,2)
Avg('y', 't', '14-Dec-97', '14 8:00')
Avg('tag1', 'tag2')
```

Badval

Test a value to see if it is bad. For real and integer points, a bad value is a digital state value. For Digital points, a bad value is a digital state value outside its own Digital State Set.

Format

```
Badval(x)
```

Arguments

```
x
```

A value to be tested.

Returns

1 if the value is bad

0 if the value is not bad

Exceptions

Returns **1** for blob points. Returns **0** for character strings.

Examples

```
BadVal('tag1')
BadVal('digitaltag')
BadVal(TagVal('stringtag', '14-Dec-97 8:00:00'))
```

Bod

Gets the timestamp for midnight at the start of a day.

Format

```
Bod(time)
```

Arguments

```
time
```

A time expression.

Returns

Timestamp for the start of the day.

Exceptions

None.

Usage Note

This function is useful for establishing a time at a unique clock time independent of the length of particular days.

Examples

```
Bod('*')
Bod('y')
Bod(FindEq('tag1', '14-Dec-97', '+17d',50))
```

See Also

Bom, Bonm, Noon

Bom

Gets the timestamp for midnight at the beginning of the month.

Format

```
Bom(time)
```

Arguments

```
time
```

A time expression.

Returns

Timestamp for the start of the month.

Exceptions

None.

Usage Note

This function is useful for establishing a time at a unique clock time independent of the length of particular days.

Examples

```
Bom('*')
Bom(PrevEvent('tag', '*'))
Bom(FindEq('tag1', '14-Dec-97', '+17d',50))
```

See Also

Bod, Bonm, Noon

Bonm

Gets the timestamp for midnight at the beginning of the next month.

Format

```
Bonm(time)
```

Arguments

```
time
```

Time expression.

Returns

Timestamp for the start of the next month.

Exceptions

None.

Usage Note

This function is useful for establishing a time at a unique clock time independent of the length of particular days.

Examples

```
Bonm('*')
Bonm('y')
Bonm(FindEq('tag1', '14-Dec-97', '+17d',50))
```

See Also

Bod, Bonm, Noon

Char

Build a string from ASCII character codes.

Format

```
Char( n1 [, n2, … ] )
```

Arguments

```
n1, n2, ...
```

Numeric numbers or expressions.

Returns

A string built from the character codes.

Exceptions

If an argument is not a number, returns an error.

Examples

```
Char(65) = "A"
Char(80, 73) = "PI"
Char(6 * 9) = "6"
```

Compare

Compare two strings with wild characters ("*" and "?").

Format

```
Compare(str1, str2 [,casesen])
```

Arguments

```
str1, str2
```

Strings. Str2 may contain wild characters.

```
casesen (optional)
```

Flag indicating if the comparison is case sensitive.

```
casesen = 0 the comparison is case insensitive (default)
casesen = 1 the comparison is case sensitive
```

Returns

1 if **Str1** = **Str2**

0 otherwise

Exceptions

Wild characters in **str1** are not treated as wild.

Examples

```
compare("?","a") = 0
compare("What","wh") = 0
compare("What","wha?") = 1
compare("What","wh*") = 1
compare("What","wh*", 1) = 0
```

Concat

Concatenate two or more strings.

Format

```
Concat(s1, s2, ..., sn)
```

Arguments

```
s1...sn
```

Must be character strings, or expressions yielding character strings.

Returns

The character strings, concatenated together. This function does not insert blanks between its arguments. If you need blanks, you must add them yourself.

Note

Consider using **Text**, which is more general and more precise in many cases than **Concat**.

Examples

```
Concat("shut", "down") = "shutdown"
```

See Also

Text

Cos

Return the trigonometric cosine of an integer or real number.

Format

```
Cos(x)
```

Arguments

```
x
```

Must be an integer or real number, which represents an angle in radians.

Returns

The cosine of **x**.

Exceptions

If **x** is not an integer or real number, returns an error value.

Examples

```
Cos('tag1')
Cos(1)
Cos(1.1)
```

See Also

Acos, Asin, Atn, Atn2, Cosh, Sin, Sinh, Tan, Tanh

Cosh

Return the hyperbolic cosine of an integer or real number.

Format

```
Cosh(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The hyperbolic cosine of **x**.

Exceptions

If **x** is not an integer or real number, returns an error value.

Examples

```
Cosh('tag1')
Cosh(.9)
Cosh(1)
```

See Also

Acos, Asin, Atn, Atn2, Cos, Sin, Sinh, Tan, Tanh

Curve

Returns the y value of a curve given the **x** value.

Format

```
Curve( x, (x1,y1) (x2,y2) … (xn,yn) )
```

Arguments

```
x
```

Expression evaluating to a number.

```
x1, y1
```

The first point on the curve. The **xi's** and **yi's** are numeric constants evaluated at compile time. The values set for **xi's** must be in ascending order.

Returns

Returns the **y** value on the curve corresponding to the value of **x.** Linear interpolation is used between points defining the curve. If the value of **x** is less than **x1** then **y1** is returned and if it is greater than **xn**, **yn** is returned. The points are assumed to be ordered in the **x** direction from smallest to largest.

Exceptions

If the value of **x** is not an integer or real number, an error value is returned.

Examples

```
curve('tag1', (0,100) (100,0) ) //inverter
curve('tag3', (25,25) (75,75) ) //limiter
```

Day

Extract the day of the month from a time expression.

Format

```
Day(time)
```

Arguments

```
time
```

A time expression.

Returns

The day of the month of time, in the range 1–31.

Exceptions

None.

Examples

```
Day('*')
Day('t')
Day(FindGt('tag1', '*-30d', '*',50))
```

See Also

DaySec, Hour, Minute, Month, Second, Weekday, Year, Yearday

DaySec

Extract the number of seconds since midnight from a time expression.

Format

```
DaySec(time)
```

Arguments

```
time
```

A time expression.

Returns

The number of seconds of time since midnight, in the range 0–86399.

Exceptions

None.

Usage Note

This function is the same as the **Time** function in the PI 2.x Performance Equation package. For example, if the current time is 8:30 am, **DaySec('*')** returns *30600*.

Examples

```
DaySec('*')
DaySec('t')
DaySec(FindGt('tag1', '*-30d', '*',50))
```

See Also

Day, Hour, Minute, Month, Second, Weekday, Year, Yearday

Delay

Delay line, the output tracks the input. For use in real time calculations, in *pipeschd.exe* for example, this function might be a better choice than Prevvalue.

Format

```
Delay( x, runflag, n )
```

Arguments

```
x
```

Must be an integer or real number.

```
runflag
```

Non-zero enables filter to run.

```
n
```

Length of the delay, integer.

Returns

The input signal delayed by **n** calculation intervals. For scan class scheduling, the calculation interval is based on the scan class. For event based scheduling, the calculation interval will be dependent on the trigger and the exception deviation.

Exceptions

Delay is not supported in the **pipetest** utility or in FactoryTalk Historian DataLink. If the input point is not a real number or integer, Delay will return an error. Delay will return **Calc Failed** until **n** scans have elapsed after startup.

Examples

```
Delay('tag1',1,2)
```

DigState

Translate a character string representing a digital state into its corresponding digital state.

Format

```
DigState(s1 [, x])
```

Arguments

```
s1
```

A character string representing a digital state.

```
x (optional)
```

A digital point in which the character string represents a digital state. If omitted, all Digital State Sets, starting with the System Digital Set, are searched for the given string.

Returns

A digital state

Exceptions

If the character string does not represent a digital state in the Digital State Set of the reference digital point, the function returns **Calc Failed**. If digital point is omitted and character string does not represent a digital state in any of the digital sets, **Calc Failed** is returned.

Examples

```
DigState("digitalstring", 'digitaltag')
StateNo(DigState("digitalstring", 'digitaltag'))
```

DigText

Obtain the text corresponding to the current digital state of a point.

Format

```
DigText(tagname)
```

Arguments

```
tagname
```

A tagname that represents a digital state variable.

Returns

The text for the digital state.

Exceptions

If the argument is not a digital state tagname, an error condition is returned.

Examples

```
DigText('alarmtag')
DigText('cdm158' )
DigText('nondigitaltag') would not compile and returns an error message
```

EventCount

Find the number of Archive events for a point over a given time.

Format

```
EventCount(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point must represent a continuous variable.

```
starttime
```

Must be a time expression representing the beginning of the time range to search.

```
endtime
```

Must be a timestamp, greater than starttime; the end of the time range to search.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must good.

Returns

Number of Archive events for the point within the specified interval.

Exceptions

If the point has no good values or the pctgood minimum is not reached for the given time range, returns an error value.

Caution

When endtime is a future time (for example, **'*+1h'**), **TagCount** might include the system digital state **No Data** and thus is larger the number of events stored in the PI Archive. Avoid using a future time if possible.

Examples

```
EventCount('tag1', 'y', '*')
EventCount('tag1', '14-Dec-97', '+1d',70)
EventCount('tag1', '14-Dec-97', '15-Dec-97')
```

Exp

Return the exponential of an integer or real number. This is the number **ex**, where **e = 2.7182818...**

Format

```
Exp(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The exponential of **x**.

Exceptions

If **x** is not an integer or real number, returns an error value.

Examples

```
Exp('tag1')
Exp(TagVal('tag1','14-Dec-97'))
Exp(11)
```

FindEq

Find the first time, within a range, when a point is equal to a given value.

Format

```
FindEq(tagname, starttime, endtime, value)
```

Arguments

`tagname`

A tagname enclosed in single quotes.

`starttime`

A time expression representing the beginning of the time range to search Relative times are relative to endtime if endtime is not itself a relative time.

`endtime`

A time expression representing the end of the time range to search. Relative times are relative to starttime if starttime is not itself a relative expression. If endtime is earlier than starttime, the range is searched backwards.

`value`

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was equal to the given value.

Exceptions

If the point was never equal to the given value, **FindEq** returns an error value.

Usage Note

FindEq interpolates between Archive events, if necessary, to find the value it is looking for.

Examples

```
FindEq('tag1', 't', '*',40.0)
FindEq('digitaltag', '-1d', '*', TagVal('digitaltag', '14-Dec-97'))
FindEq('digitaltag', '14-Dec-97', '*', "On")
```

FindGE

Find the first or last time, within a range, when a point is greater than or equal to a given value.

Format

```
FindGE(tagname, starttime, endtime, value)
```

Arguments

`tagname`

A tagname.

`starttime`

A time expression representing the beginning of the time range to search or a time relative to endtime, if endtime is a time.

`endtime`

A time expression representing the end of the time range to or a time (in seconds) relative to starttime, if starttime is a time. If endtime is earlier than starttime, the range is searched backwards.

`value`

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was greater than or equal to the given value.

Exceptions

If the point was always less than the given value, **FindGE** returns an error value.

Usage Note

FindGE interpolates between archive events, if necessary, to find the value it is looking for.

Examples

```
FindGE('tag1', 't', '*',40.0)
FindGE('digitaltag', '-1d', '*', TagVal('digitaltag', '14-Dec-97'))
FindGE('tag1', '-1d', '*','tag2')
```

FindGT

Find the first time, within a range, when a point is greater than a given value.

Format

```
FindGT(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

A time expression representing the beginning of the time range to search. Can be a time relative to endtime if endtime is a time.

```
endtime
```

End of the time range to search, time expression or time (in seconds) relative to starttime if starttime is a time. If this time is earlier than starttime, the range is searched backwards.

```
value
```

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was greater than the given value.

Exceptions

If the point was never greater than the given value, **FindGT** returns an error value.

Usage Note

FindGT interpolates between Archive events, if necessary, to find the value it is looking for.

Examples

```
FindGT('tag1', 't', '*',40.0)
FindGT('tag1', '-1d', '*',40.0)
FindGT('digitaltag', '-1d', '*', TagVal('digitaltag', 'y'))
```

FindLE

Find the first time, within a range, when a point is less than or equal to a given value.

Format

```
FindLE(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search; time expression or time relative to endtime if endtime is a time.

```
endtime
```

End of the time range to search, timestamp or time (in seconds) relative to starttime if starttime is a time. If this time is earlier than starttime, the range is searched backwards.

```
value
```

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was less than or equal to the given value.

Exceptions

If the point was always greater than the given value, **FindLE** returns an error value.

Usage Note

FindLE interpolates between Archive events, if necessary, to find the value it is looking for.

Examples

```
FindLE('tag1', 't', '*',40.0)
FindLE('tag1', -3600, '*',40.0)
FindLE('tag1', 'Saturday', '*',40.0)
```

FindLT

Find the first time, within a range, when a point is less than a given value.

Format

```
FindLT(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search; time expression or time relative to endtime if endtime is a time.

```
endtime
```

End of the time range to search, time expression or time (in seconds) relative to starttime if starttime is a time. If this time is earlier than starttime, the range is searched backwards.

```
value
```

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was less than the given value.

Exceptions

If the point was never less than the given value, **FindLT** returns an error value.

Usage Note

FindLT interpolates between Archive events, if necessary, to find the value it is looking for.

Examples

```
FindLT('tag1', 't', 3600,40.0)
FindLT('tag1', -1h, '*',40.0)
FindLT('tag1', '14-Dec-97 01:00:00.0001, '*',40.0)
```

FindNE

Find the first time, within a range, when a point is unequal to a given value.

Format

```
FindNE(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search; time expression or time relative to endtime if endtime is a timestamp.

```
endtime
```

End of the time range to search, time expression or time (in seconds) relative to starttime if starttime is a time. If this time is earlier than starttime, the range is searched backwards.

```
value
```

Must be an integer or real number or digital state (character string), the value to search for.

Returns

The timestamp closest to starttime, within the given range, for which the point was unequal to the given value.

Exceptions

If the point was always equal to the given value, **FindNE** returns an error value.

Examples

```
FindNE('tag1', 'y', '*',40.0)
FindNE('tag1', '14-Dec-97', '*',40.0)
FindNE('tag1', '14-Dec-97', 'Monday',40.0)
```

Float

Convert a string to a number.

Format

```
Float(x)
```

Arguments

```
x
```

A string or number.

Returns

A number for a numeric string and Calc Failed for a non-numeric string. If **x** is already a number, **x** is returned.

Examples

```
Float(12.3)  = 12.3
Float('sinusoid')
Float("-12.3")  = -12.3
```

Format

Convert a number to string according to a format expression.

Format

```
Format(num, format [,num_type ])
```

Arguments

```
num
```

A number (real or integer).

```
format
```

Format-control string. This is the same as that used by the C language function **Sprintf**.

```
num_type (optional)
```

Number-type character. This must be either R(eal) or I(nteger). The default is **R**.

Returns

A formatted string.

Examples

```
Format('sinusoid', "%3.3f", "R") = "66.890"
Format(45, "%3.3d") = "045"
Format(45, "%3.3d", "I") = "045"
Format(45, "%3.3d", "R") = "000" (Don't do this!)
```

Frac

Returns the fractional part of a real number. Returns **0** for integers.

Format

```
Frac(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The fractional part of **x**.

Exceptions

If **x** is not an integer or real number, returns an error value.

Usage Note

By definition: *Int(*x*)* + *Frac(*x*)* = x.

Examples

```
Frac('tag1')
Frac(1.1)
Frac(TagVal('tag1', '14-Dec97'))
```

Hour

Extract the hour from a time expression.

Format

```
Hour(time)
```

Arguments

```
time
```

A time expression.

Returns

The hour of time, in the range 0–23.

Exceptions

None.

Examples

```
Hour('*')
Hour('Saturday')
Hour('t')
```

See Also

Day, DaySec, Minute, Month, Second, Weekday, Year, Yearday

Impulse

Dynamic response specified by the impulse response.

Format

```
Impulse(tagname, runflag, i1,i2 … )
```

Arguments

```
tagname
```

Must be a tagname for a numerical point.

```
runflag
```

Non-zero enables filter to run.

```
i1, i2, …
```

Unit impulse response specifying dynamic model, text sequence of numbers.

Returns

Dynamic model output as function of time.

```
u(t)=i1*u(t-1) + i2*u(t-2) + …
```

Where u(t) is the current output and u(t-1) is the output one sample interval in the past.

Exceptions

Impulse gives different results depending on which type of scheduling is used. In clock scheduling, the interval between time series values depends on the scan class and gives values at evenly spaced time intervals.

On the other hand, event-based scheduling is dependent on a trigger from another point. If the exception deviation is not zero, the intervals for events are not evenly spaced in time—hence **Impulse** gives results that are not trustworthy. Impulse is not supported in the **pipetest** utility or in FactoryTalk Historian DataLink. If the input point is not a real number or integer, **Impulse** returns an error.

Examples

```
Impulse('tag1',1,1,1,1)
```

InStr

Determine the location within a string where a sub-string match is first found.

Format

```
InStr([start,] str1, str2 [,casesen])
```

Arguments

```
start (optional)
```

An integer specifying which character in **str1** to start the comparison. Must be larger than or equal to **0**.

```
str1, str2
```

Two strings and/or points with string pointtypes to be compared.

```
casesen (optional)
```

Flag indicating if the comparison is case sensitive.

`casesen = 0` the comparison is case insensitive (default)

`casesen = 1` the comparison is case sensitive

Returns

0 if **str2** is not a sub-string of **str1** starting from the start position; otherwise, the location of character where **str2** first matches the characters in **str1** from the start position.

Exceptions

Wild characters are not treated as wild.

Examples

```
InStr("What", "At") = 3
InStr("What What What", "What") = 1
InStr("what", "At", 1) = 0
InStr(4,"what","At") = 0
InStr('StringTag', "Error") = 1 (if the tag value for 'stringtag' is "Error")
InStr('StringTag',"StringTag") = 0 (if the tag value for 'stringtag' is
"Error")
```

Int

Return the integer part of an integer or real number.

Format

```
Int(x)
```

Arguments

```
x
```

A number or string.

Returns

The integer part of **x.** If **x** is a string, it is first converted into a number.

Exceptions

If **x** is not a number or a numeric string, returns **Calc Failed**.

Examples

```
Int('tag1')
Int(1)
Int(2.1)
Int("2.1")
```

IsDST

Determine if a time expression is in a daylight saving time (DST) period on the local machine.

Format

```
IsDST(time)
```

Arguments

```
time
```

A time expression.

Returns

1 if the time is in a DST period and **0** otherwise.

Exceptions

If the argument is not a time value, an error condition is returned.

Examples

```
IsDST('*')
IsDST('*-182.5d')
IsDST('t')
IsDST('timestringtag')
```

IsSet

Determine if a PI value is annotated, substituted, or questionable.

Format

```
IsSet(pivalue, select)
```

Arguments

`pivalue`

Any PI value. May be an integer, real number, digital state, or character string.

select

A string but only the first character is considered. "a" for annotated; "s" for substituted; and "q" for questionable. It is case-insensitive.

Returns

1 if true and **0** otherwise.

Exceptions

None.

Examples

```
IsSet('sinusoid', "a")
IsSet('sinusoid', "annotated")
IsSet('sinusoid', "annotatted is mispelled")
IsSet('stringtag',"annotatiiion is mispelled but it does not matter.")
IsSet('stringtag',"A")
IsSet('alarmtag1',"q")
IsSet('stringtag',"s")
```

LCase

Convert a string to a lowercase string.

Format

```
LCase(strexp)
```

Arguments

```
strexp
```

Must be a string.

Returns

A string that has been converted to lowercase.

Exceptions

If the argument is not a string, returns an error value.

Examples

```
LCase("Stringtag") = "stringtag"
LCase('Stringtag') = "error" if the Snapshot value for the stringtag equals
"Error"
```

See Also

UCase

Left

Determine a specified number of characters of a string from the left.

Format

```
Left(str, len)
```

Arguments

```
str
```

A string.

```
len
```

An integer.

Returns

len characters of the string from the left.

Exceptions

If the arguments are not of the required types, returns an error.

Examples

```
Left("Stringtag", 3) = "Str"
Left('Stringtag', 3) = "Err" if the Snapshot value for the stringtag equals
"Error"
```

See Also

Mid, Right

Len

Determine the length of a string.

Format

```
Len(str)
```

Arguments

```
str
```

A string.

Returns

The length of a string.

Exceptions

If the argument is not a string, returns an error value.

Examples

```
Len("Stringtag") = 9
Len('Stringtag') = 5 if the Snapshot value for the stringtag equals "Error"
```

Log

Return the natural (base-e = 2.7182818...) logarithm of an integer or real number.

Format

```
Log(x)
```

Arguments

```
x
```

Must be an integer or real number greater than zero.

Returns

The natural logarithm of x.

Exceptions

If **x** is zero or negative, or not a number, returns an error value.

Examples

```
Log('*')
Log(14)
Log(TagVal('tag1', '14-Dec-97'))
```

See Also

Log10

Log10

Return the common (base-10) logarithm of an integer or real number.

Format

```
Log10(x)
```

Arguments

```
x
```

Must be an integer or real number greater than zero.

Returns

The common logarithm of **x**.

Exceptions

If **x** is zero or negative, or not a number, returns an error value.

Examples

```
Log10('*')
Log10(14)
Log10(TagVal('tag1', '14-Dec-97'))
```

See Also

Log

LTrim

Remove the leading blanks from a string.

Format

```
LTrim(str)
```

Arguments

```
str
```

A string.

Returns

A string with leading blanks removed.

Exceptions

If str is not a string, an error value is returned.

Examples

```
LTrim(" Stringtag") = "Stringtag"
LTrim("Stringtag ") = "Stringtag "
LTrim('Stringtag') = "Error" if the Snapshot value for the stringtag equals "
Error"
```

See Also

RTrim, Trim

Max

Return the maximum of arguments.

Format

```
Max(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be numbers, times, or time periods, but all must be the same.

Returns

The maximum of the arguments. The result has the same data type as the arguments.

Exceptions

Arguments whose run-time values are digital states are ignored. If all values are digital states, **Max** returns an error value.

Examples

```
Max('*', 'y', 'Saturday')
Max(14, 'tag1', 14.5, TagVal('tag2','14-Dec-97'))
Max('*'-'*-h', 't'-'y', TimeEq('tag1', 'y', 't',50))
```

See Also

Min

Median

Return the median (middle) value of three or more arguments.

Format

```
Median(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be only integers, real numbers, times, or time periods, but all arguments must be the same data type.

Returns

The median value of the input arguments. If the number of arguments is even, the average of the two middle values is returned.

Exceptions

Arguments whose run-time values are digital states are ignored. The function must have greater than two arguments that evaluate to non-digital states; otherwise, **Median** returns an error value.

Usage Note

Median allows for mixed integer and real data types. **Median** follows the data type of the first argument. Hence if the first argument is a point that evaluates to an integer then all the other entries will be converted to integers by truncation (not by rounding).

Examples

```
Median('*', 'y', 'Saturday')
Median(14, 'tag1', 14.5, TagVal('tag2','14-Dec-97'))
Median('*'-'*-1h', 't'-'y', TimeEq('tag1', 'y', 't',50))
```

MedianFilt

Return the median value of the last specified number of values of a time series.

Format

```
MedianFilt( tagname, runflag, number )
```

Arguments

```
tagname
```

Must be a numerical point.

```
runflag
```

Non-zero enables filter to run.

```
number
```

The number of series elements to be considered. A numeric constant greater than or equal to **3**.

Returns

The median value of the last number values in the series of values.

Exceptions

Arguments whose run-time values are digital states are ignored. **MedianFilt** is not supported in the **pipetest** utility or in FactoryTalk Historian DataLink. If all values are digital states, **MedianFilt** returns an error value.

Examples

```
MedianFilt('tag1',1,3)
```

Mid

Return a sub-string within a string.

Format

```
Mid(str, start [,len])
```

Arguments

```
str
```

A string.

```
start
```

An integer specifying the position of the first character within the string. The first character in the string is number **1**.

```
len (optional)
```

The maximum length of the returned string. The default is the length of the string.

Returns

len characters of the string to the left of (and including) the first character whose position is specified by start.

Exceptions

If the arguments are not of the required types, an error value is returned. The maximum number of characters that can be returned is 999.

Examples

```
Mid("Stringtag", 3) = "ringtag"
Mid("Stringtag", 3, 2) = "ri"
Mid('Stringtag', 1, 1) = "E" if the Snapshot value for the stringtag equals
"Error"
```

See Also

Left, Right

Min

Return the minimum of arguments.

Format

```
Min(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be numbers, times, or time periods, but all must be the same data type.

Returns

The minimum of the arguments. The result has the same data type as the arguments.

Exceptions

Arguments whose run-time values are digital states are ignored. If all values are digital states, **Min** returns an error value.

Examples

```
Min('*', 'y', 'Saturday')
Min(14, 'tag1', 14.5, TagVal('tag2','14-Dec-97'))
Min('*'-'*-1h', 't'-'y', TimeEq('tag1', 'y', 't',50))
```

See Also

Max

Minute

Extract the minute from a time expression.

Format

```
Minute(time)
```

Arguments

```
time
```

A time expression.

Returns

The minute of time, in the range 0–59.

Exceptions

None.

Examples

```
Minute('*')
Minute('1')
Minute('*-1h')
```

See Also

Day, DaySec, Hour, Month, Second, Weekday, Year, Yearday

Month

Extract the month from a time expression.

Format

```
Month(time)
```

Arguments

```
time
```

A time expression.

Returns

The month of time, in the range 1–12.

Exceptions

None.

Examples

```
Month('*')
Month('1')
Month('*-1h')
```

See Also

Day, DaySec, Hour, Minute, Second, Weekday, Year, Yearday

NextEvent

Find the time of a point's next Archive event after a given time.

Format

```
NextEvent(tagname, time)
```

Arguments

```
tagname
```

A tagname.

time

A time expression.

Returns

The timestamp of the next Archive event for tagname after time.

Exceptions

If point has no Archive data after time, returns an error value.

Examples

```
NextEvent('tag1','*')
NextEvent('digitaltag', '*')
```

See Also

NextVal, PrevEvent, PrevVal, TagVal

NextVal

Find the value of a point's next Archive event after a given time.

Format

```
NextVal(tagname, time)
```

Arguments

```
tagname
```

A tagname.

time

A time expression.

Returns

The value of the next Archive event for tagname after time.

Exceptions

If point has no Archive data after time, returns an error value.

Examples

```
NextVal('tag1','*-1h')
NextVal('digitaltag', '14-Dec-97')
```

See Also

NextEvent, PrevEvent, PrevVal, TagVal

Noon

A timestamp for noon on the day of a given time expression.

Format

```
Noon(time)
```

Arguments

```
time
```

A time expression.

Returns

A timestamp corresponding to noon of the day of the input time.

Exceptions

None.

Usage Note

This function is useful for establishing a unique clock time independent of the length of particular days.

Examples

```
Noon('*')
Noon('14-Dec-97')
```

See Also

Bod, Bom, Bonm

NoOutput

Do not send the current calculation result to PI.

Format

```
NoOutput()
```

Arguments

```
None
```

Usage Note

It is important to include the parentheses after this function (use **NoOutput()** instead of **NoOutput** as **NoOutput** is an invalid syntax). This function applies only to the current calculation. The output of this function in **pipetest**.*exe* is "**NoOutput() Called**".

Example

```
If 'PITag' < 100 or 'PItag' > 0 then 'PITag' else NoOutput()
```

ParseTime

Translate a PI time expression to a timestamp.

Format

```
ParseTime(s)
```

Arguments

```
s
```

Must be a character string in PI time format.

Returns

The timestamp corresponding to **s**.

Exceptions

If **s** is not a character string, or if there is a syntax error, returns an error value.

Usage Note

There is no difference between **ParseTime**("14-Nov-92") and the time expression '14-Nov-92', except that the **ParseTime** call definitely takes more time. This is because the time expression (enclosed in single quotes) is evaluated at compile time, not run time.

If you write **ParseTime**('14-Nov-92') (using single quotes, not double quotes) the parser will detect an error, because the expression in single quotes is already translated to a timestamp at compile time.

The expression **ParseTime**(":12:00:00") is not the same as the time expression ':12:00:00'. The **ParseTime** expression is evaluated at runtime and translated using '*' as the relative time base, while the time expression is evaluated at compile time and uses the time the expression is parsed as the relative time base.

Examples

```
ParseTime(”14-Dec-97”)
ParseTime(”t”)
```

PctGood

Find the time percentage, over a given range, when a point's archived values are good.

Format

```
PctGood(tagname, starttime, endtime)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Must be a time expression, the beginning of the time range to search.

```
endtime
```

Must be a time expression, greater than starttime; the end of the time range to search.

Returns

An integer or real number from 0.0 to 100.0: the percentage of the given time when the point had good values.

Examples

```
PctGood('tag1', 'y','*')
PctGood('tag1', '-1h', '*')
```

Poly

Evaluate the polynomial **c0 + c1x + c2x2 + … +cnxn**.

Format

```
Poly(x, c0, ..., cn)
```

Arguments

```
x
```

The variable. It must be an integer or real number.

```
c0...cn
```

The coefficients. There must be at least one coefficient. All must be numbers.

Returns

The value of the polynomial.

Exceptions

If **x** or any coefficient is not an integer or real number, Poly returns an error value.

Examples

```
Poly('tag1',1,1)
```

PrevEvent

Find the time of a point's previous Archive event before a given time.

Format

```
PrevEvent(tagname, time)
```

Arguments

```
tagname
```

A tagname.

```
time
```

A time expression.

Returns

The timestamp of the previous Archive event for tagname before time.

Exceptions

If point has no Archive data before time, returns an error value.

Examples

```
PrevEvent('tag1', '*')
PrevEvent('tag1','14-Dec-97')
```

See Also

NextEvent, NextVal, PrevVal, TagVal

PrevVal

Find the value of a point's previous Archive event before a given time.

Format

```
PrevVal(tagname, time)
```

Arguments

```
tagname
```

A tagname.

```
time
```

A time expression.

Returns

The value of the previous Archive event for tagname before time.

Exceptions

If point has no Archive data before time, returns an error value.

Examples

```
PrevVal('tag1', '*')
PrevVal('tag1','14-Dec-97')
```

See Also

NextEvent, NextVal, PrevEvent, TagVal

PStDev

Return the standard deviation of two or more arguments, where those arguments represent the whole population. The standard deviation of a population **$x_1 \ldots x_n$** is

$$\sqrt{\frac{\sum (x_i - \mu)^2}{n}}$$

where μ is the mean of the arguments, i.e.,

$$\frac{\sum x_i}{n}$$

Format

```
PStDev(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be numbers or time expressions, but all must be the same.

Returns

The standard deviation of the arguments. If the arguments are numbers, the result is a number; if the arguments are times or time periods, the result is a time period.

Exceptions

Arguments whose run-time values are digital states are ignored. If all values are digital states, **PStDev** returns an error value.

Usage Note

In most cases you should use Sstdev instead of **PstDev**. **Sstdev** calculates the standard deviation of a sample.

Examples

```
PStDev('tag1', 'tag2')
PStDev('*','14-Dec-97', 'y')
PStDev('*'-'y','14-Dec-97'-'*', '-1h')
```

See Also

SStDev

Range

Find the difference between a point's maximum and minimum values during a given time, according to values stored in the PI Archive.

Format

```
Range(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point should represent a continuous variable.

```
starttime
```

Must be a time expression, the beginning of the time range to search.

```
endtime
```

Must be a time expression, greater than starttime; the end of the time range to search.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must good.

Returns

The difference between the point's maximum and minimum values during the given time.

Exceptions

If the point has no good values or the pctgood minimum is not reached in the given time range, returns an error value.

Caution

The **OverRangeStat** and **UnderRangeStat** digital states are not taken into account when calculating this value.

Examples

```
Range('tag1', 'y', '*')
Range('tag1','-1h', 'y')
Range('tag1','y', '+1h',70)
```

Right

Determine a specified number of characters of a string from the right.

Format

```
Right(str, len)
```

Arguments

```
str
```

A string.

```
len
```

An integer.

Returns

len characters of the string from the right.

Exceptions

If the arguments are not of the required types, an error value is returned.

Examples

```
Right("Stringtag", 3) = "tag"
Right('Stringtag', 4) = "rror" if the Snapshot value for the stringtag equals
"Error"
Right("Stringtag", 20) = "Stringtag"
```

See Also

Left, Mid

Round

Round a number or time to the nearest unit.

Format

```
Round(x [, unit])
```

Arguments

```
x
```

Must be an integer or real number or time expression.

```
unit (optional)
```

The size of the unit to round to. If **x** is a number, unit must be a number. If **x** is a time expression or time period, unit must be a time period. If unit is omitted, **Round** rounds to the nearest integer (for a number) or second (for a time period).

Returns

The nearest value to **x** which is an integer multiple of unit. Returns the same data type as **x.** For more information, see the examples below.

Exceptions

If **x** is a string, or if unit is of the wrong data type, returns an error value.

Examples

Expression	**Value**	**Comments**
Round(12.499)	12.0	Round to nearest integer
Round(12.500)	13.0	Half a unit rounds up
Round(12.8, 10)	10.0	Round to nearest ten
Round('14-Dec-97 11:47, '+1h')	14-Dec-97 12:00	Round to nearest hour (returns timestamp)
Round('18:47' –'15:00','+1h')	10800	Round period to nearest hour (returns period in seconds)

Note: *Round* to the nearest day results in a timestamp of the closest day in UTC time and not local time.

Usage Note

If **x** is time and unit is omitted this routine has no effect: times are accurate only to 1 second.

See Also

Trunc

RTrim

Trim trailing blanks from a string.

Format

```
RTrim(str)
```

Arguments

```
str
```

A string.

Returns

The source string with trailing blanks removed.

Exceptions

If **str** is not a string, an error value is returned.

Examples

```
RTrim("Stringtag ") = "Stringtag" "
RTrim(" Stringtag") = " Stringtag"
RTrim('Stringtag') = "Error" if the Snapshot value for the stringtag equals
"Error "
```

See Also

LTrim, Trim

Second

Extract the second from a time expression.

Format

```
Second(time)
```

Arguments

```
time
```

A time expression.

Returns

The second of time, in the range 0–59.

Exceptions

None.

Examples

```
Second('*')
Second('y')
Second('*-1h')
```

See Also

Day, DaySec, Hour, Minute, Month, Weekday, Year, Yearday

Sgn

Return a representation of the numerical sign of a number.

Format

```
Sgn(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

-1 if x < 0.
0 if x = 0.
1 if x > 0.

Exceptions

If **x** is not an integer or real number, returns an error value.

Examples

```
Sgn('tag1')
Sgn(1)
Sgn(0)
```

Sin

Return the trigonometric sine of a number.

Format

```
Sin(x)
```

Arguments

```
x
```

Must be an integer or real number, which represents an angle in radians.

Returns

The sine of **x**.

Exceptions

If **x** is not a number, returns an error value.

Examples

```
Sin('tag1')
Sin(1)
Sin(1.1)
```

See Also

Acos, Asin, Atn, Atn2, Cos, Cosh, Sinh, Tan, Tanh

Sinh

Return the hyperbolic sine of a number.

Format

```
Sinh(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The hyperbolic sine of **x**.

Exceptions

If **x** is not a number, returns an error value.

Examples

```
Sinh('tag1')
Sinh(1)
Sinh(0.9)
```

See Also

Acos, Asin, Atn, Atn2, Cos, Cosh, Sin, Tan, Tanh

Sqr

Return the square root of a number.

Format

```
Sqr(x)
```

Arguments

```
x
```

Must be an integer or real number greater than or equal to zero.

Returns

The square root of **x**.

Exceptions

If **x** is negative, or is not a number, returns an error value.

Examples

```
Sqr('tag1')
Sqr(2)
Sqr(2.1)
```

SStDev

Return the standard deviation of two or more arguments, where those arguments represent a sample of a larger population. The standard deviation of a sample $\mathbf{x}_1...\mathbf{x}_n$ is equal to

$$\sqrt{\frac{\sum (x_i - \mu)^2}{n-1}}$$

Where μ is the sample mean, i.e.,

$$\frac{\sum x_i}{n}$$

Format

```
SStDev(x1, x2, ..., xn)
```

Arguments

$x_1...x_n$

May be numbers or time expressions, but all must be the same.

Returns

The sample standard deviation of the arguments. If the arguments are numbers, the result is a number; if they are times or time periods, the result is a time period (number of seconds).

Exceptions

Arguments whose run-time values are digital states are ignored. If there are not at least two numeric values, **SStDev** returns a zero.

Usage Note

In the rare case where you have the entire population, rather than a sample, you might use the function **PstDev**, rather than **SStDev**.

Examples

```
SStDev('tag1', 'tag2', TagVal('tag1', 'y'))
SStDev('y', 't', '14-Dec-97')
SStDev(1, 2, 1.1)
```

See Also

PStDev

StateNo

Translate a digital state into its corresponding state number.

Format

```
StateNo(digstate)
```

Arguments

```
digstate
```

A digital state value.

Returns

The offset into the Digital State Set corresponding to **digstate**.

Exceptions

If a point is passed as **digstate** that is not a digital point, returns an error value.

Usage Note

A digital state may appear more than once in the digital state Table. In this case, the value that **StateNo** returns may vary. If **digstate** is the value of a digital point, **StateNo** returns a code number appropriate for that point.

Examples

```
StateNo('digitaltag')
StateNo(TagVal('digitaltag', '*-1h'))
```

StDev

Find the time-weighted standard deviation of a point over a given time, according to values stored in the PI Archive.

Format

```
StDev(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point must represent a continuous variable.

```
starttime
```

Must be a time expression representing the beginning of the time range to search.

```
endtime
```

Must be a time expression, greater than starttime; representing the end of the time range to search.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must be good.

Returns

The point's time-weighted standard deviation over the given time.

Exceptions

If the point has no good values or the **PctGood** minimum is not reached for the given time range, returns an error value.

Caution

If the point has few good Archive values during the time period, this function's result may not be trustworthy. Use the **PctGood** function to find out what percentage of the values is good.

Examples

```
StDev('tag1', 'y', '*')
StDev('tag1', '14-Dec-97', '+1d',85)
StDev('tag1', '14-Dec-97', '15-Dec-97')
```

See Also

PctGood

String

Convert any value to a string.

Format

```
String(anyvalue)
```

Arguments

```
anyvalue
```

Any expression. It may be of any of the normal PI System data types.

Returns

The string representing the value argument.

Exceptions

None.

Examples

```
String(12.23) = "12.23"
String('sinusoid')
String('pidigital')
String('*')
String("Hello, PI user!") = "Hello, PI user! "
```

TagAvg

Find the time-weighted average value of a point over a given time, according to values stored in the PI Archive.

Format

```
TagAvg(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point must represent a continuous variable.

```
starttime
```

Must be a time expression representing the beginning of the time range to search.

```
endtime
```

Must be a time expression, greater than starttime; representing the end of the time range to search.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must be good.

Returns

The point's time-weighted average value over the given time.

Exceptions

If the point has no good values or the pctgood minimum is not reached for the given time range, returns an error value.

Caution

If the point has few good Archive values during the time period, this function's result may not be trustworthy. Use the **PctGood** function to find out what percentage of the values are good.

Examples

```
TagAvg('tag1', 'y', '*')
TagAvg('tag1', '14-Dec-97', '+1d',70)
TagAvg('tag1', '14-Dec-97', '15-Dec-97')
```

See Also

PctGood

TagBad

Test if a point has an abnormal state at a given time. If the point's type is **R** or **I**, any digital state is abnormal. If the point is type **D,** the states that are defined for that point are normal; all others are abnormal.

Format

```
Tagbad(tagname [, time])
```

Arguments

```
tagname
```

A tagname.

```
time (optional)
```

A time expression. If omitted, the current time ('*') is used.

Returns

0 if the point's state at time is normal, **1** if it is abnormal.

Exceptions

If point does not exist, or has no archived value at time, returns an error value.

Usage Note

Badval can test any value or expression; **TagBad** can only test a point.

Examples

```
TagBad('tag1', '*')
TagBad('digitaltag', '14-Dec-97')
TagBad('tag1', 'y')
```

See Also

Badval

TagDesc

Get a point's descriptor from the Point Database.

Format

```
TagDesc(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's descriptor.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagDesc('tag1')
TagDesc('digitaltag')
```

TagEU

Get a point's engineering unit string from the Point Database.

Format

```
TagEU(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's engineering units.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagEU('tag1')
```

TagExDesc

Get a point's extended descriptor from the Point Database.

Format

```
TagExDesc(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's extended descriptor.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagExDesc('tag1')
```

TagMax

Find the maximum value of a point during a given time, according to values stored in the PI Archive.

Format

```
TagMax(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

A time expression indicating the beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time. For example:

```
TagMax('tag1', '-1h', '*',95)
```

Here, the starttime is one hour before the endtime, which is now ('*').

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must be good.

Returns

The point's maximum value during the given time.

Exceptions

If the point has no good values or the pctgood minimum is not reached for the given time range, returns an error value.

Caution

The **OverRange** digital state is not taken into account when calculating this value.

Examples

```
TagMax('tag1', 'y', '*')
TagMax('tag1', '-1h', '*',95)
TagMax('tag1', '14-Dec-97', '+1h')
```

TagMean

Find the average value of a point over a given time, according to values stored in the PI Archive.

Format

```
TagMean(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point must represent a continuous variable.

```
starttime
```

Must be a time expression representing the beginning of the time range to search.

```
endtime
```

Must be a time expression, greater than starttime; representing the end of the time range to search.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must good.

Returns

The point's average value over the given time. Notice that the average is not time-weighted.

Exceptions

If the point has no good values or the pctgood minimum is not reached for the given time range, returns an error value. Unlike some other summary functions, **TagMean** does not interpolate any value on the boundary. Thus, if there is no Archive event between the specified interval, an error value is returned.

Caution

If the point has few good Archive values during the time period, this function's result may not be trustworthy. Use the **PctGood** function to find out what percentage of the values is good.

Examples

```
TagMean('tag1', 'y', '*')
TagMean('tag1', '14-Dec-97', '+1d',70)
TagMean('tag1', '14-Dec-97', '15-Dec-97')
```

See Also

PctGood

TagMin

Find the minimum value of a point during a given time, according to values stored in the PI Archive.

Format

```
TagMin(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point should represent a continuous variable.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must good.

Returns

The point's minimum value during the given time.

Exceptions

If the point has no good values or the pctgood minimum is not reached for the given time range, returns an error value.

Caution

The **UnderRange** digital state is not taken into account when calculating this value.

Examples

```
TagMin('tag1', 'y', '*')
TagMin('tag1', '-1h', '*',90)
TagMin('tag1', '14-Dec-97', '+1h')
```

TagName

Get a point's name from the Point Database.

Format

```
TagName(tag)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's name.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagName('tag1')
```

TagNum

Get a point's number from the Point Database.

Format

```
TagNum(string)
```

Arguments

```
string
```

A tagname in double quotes.

Returns

The point's number.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagNum("tag1")
```

TagSource

Get a point's point source character from the Point Database.

Format

```
TagSource(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's point source character.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagSource('tag1')
```

TagSpan

Get a point's span from the Point Database.

Format

```
TagSpan(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's span. If the point's type is Digital this is an integer whose value is the number of digital states defined for the point.

Examples

```
TagSpan('tag1')
TagSpan('digitaltag')
```

TagTot

Find the totalized value (time integral) of a point over a given time, according to values stored in the PI Archive.

Format

```
TagTot(tagname, starttime, endtime [, pctgood])
```

Arguments

```
tagname
```

A tagname. This point must represent a continuous process flow.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time.

```
pctgood (optional)
```

Minimum time percentage over the given time range, that the point's archived values must be good.

Returns

The point's totalized value over the given time.

Exceptions

If the point has no good values or the **PctGood** minimum is not reached for the given time range, returns an error value.

Caution

If the point has few good Archive values during the time period, this function's result may not be trustworthy. Use the **PctGood** function to find out what percentage of the value is good.

Usage Note

The system chooses a scale factor such that the integral will be correct only if the flow is expressed in units per day. If the flow is expressed in units per hour, or per some other time unit, you must multiply this result by a conversion factor. The conversion factor equals the number of actual flow time units in a day.

For instance, if you totalize a point measured in gallons per minute, you must multiply the result of **TagTot** by 1440 to get the answer in gallons. This conversion factor is not related to the time period you are totalizing over; it is strictly a function of the point's engineering units.

Some PI sites have the default total period configured to be per-hour rather than per-day. If you are at one of these sites, your conversion factor will differ.

Examples

```
TagTot('tag1', 'y', '*')
TagTot('tag1', '-1h', '*',85)
TagTot('tag1', '14-Dec-97', '+1h')
```

See Also

PctGood

TagType

Get a point's type character (I, R, or D) from the Point Database.

Format

```
TagType(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's type character.

Exceptions

If point does not exist, returns an error value.

Examples

```
TagType('tag1')
TagType('digitaltag')
```

TagTypVal

Get a point's typical value from the Point Database.

Format

```
TagTypVal(tagname)
```

Arguments

tagname

`A tagname.`

Returns

The point's typical value. If the point's type is **R** or **I,** this is a number; if the point's type is **D**, this is a digital state (character string).

Exceptions

If point does not exist, returns an error value.

Examples

```
TagTypVal('tag1')
TagTypVal('digitaltag')
```

TagVal

Find a point's Archive value at a given time.

Format

```
TagVal(tagname [, time])
```

Arguments

```
tagname
```

A tagname.

```
time (optional)
```

A time expression. If you omit this argument, '*' is used.

Returns

The archived value of tagname at time. This value is interpolated unless the point has resolution code 4.

Exceptions

If point does not exist, or has no archived value at time, returns an error value.

Examples

```
TagVal('tag1')
TagVal('digitaltag')
TagVal('tag1','*')
```

See Also

NextEvent, NextVal, PrevEvent, PrevVal

TagZero

Get a point's zero value from the Point Database.

Format

```
TagZero(tagname)
```

Arguments

```
tagname
```

A tagname.

Returns

The point's zero value. If the point's type is **R** or **I**, this is a number; if the point's type is **D**, this is a digital state (character string).

Exceptions

If point does not exist, returns an error value.

Examples

```
TagZero('tag1')
TagZero('digitaltag')
```

Tan

Return the trigonometric tangent of a number.

Format

```
Tan(x)
```

Arguments

```
x
```

Must be an integer or real number, which represents an angle in radians.

Returns

The tangent of **x**.

Exceptions

If **x** is not a number, returns an error value.

Examples

```
Tan('tag1')
Tan(1)
Tan(1.1)
```

See Also

Acos, Asin, Atn, Atn2, Cos, Cosh, Sin, Sinh, Tanh

Tanh

Return the hyperbolic tangent of a number.

Format

```
Tanh(x)
```

Arguments

```
x
```

Must be an integer or real number.

Returns

The hyperbolic tangent of **x**.

Exceptions

If **x** is not a number, returns an error value.

Examples

```
Tanh('tag1')
Tanh(1)
Tanh(1.1)
```

See Also

Acos, Asin, Atn, Atn2, Cos, Cosh, Sin, Sinh, Tan

Text

Concatenate strings representing argument values.

Format

```
Text(val1 [, val2, … ])
```

Arguments

```
val1, val2, …
```

Any expression. These may be of any of the normal PI System data types.

Returns

A string that is the concatenation of strings representing the argument values.

Examples

```
Text('sinusoid')
Text("The value for tag sinusoid is at ", '*', " is ", 'sinusoid') = "The value
for tag sinusoid at 1-Jun-00 17:07:18 is 89.09"
```

See Also

Concat

TimeEq

Find the total time, within a range, when a point is equal to a given value.

Format

```
TimeEq(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
value
```

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was exactly equal to the given value.

Exceptions

None.

Examples

```
TimeEq('tag1', 't', '*',40.0)
TimeEq('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeEq('digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeGE, TimeGT, TimeLE, TimeLT, TimeNE

TimeGE

Find the total time, within a range, when a point is greater than or equal to a given value.

Format

```
TimeGE(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
value
```

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was greater than or equal to the given value.

Exceptions

None.

Usage Note

TimeGE interpolates between Archive events, if necessary, to find the times when the point crossed the given value.

Examples

```
TimeGE('tag1', 't', '*',40.0)
TimeGE('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeGE('digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeEq, TimeGT, TimeLE, TimeLT, TimeNE

TimeGT

Find the total time, within a range, when a point is greater than a given value.

Format

```
TimeGT(tagname, starttime, endtime, value)
```

Arguments

`tagname`

A tagname.

`starttime`

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

`endtime`

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

`value`

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was greater than the given value.

Exceptions

None.

Usage Note

TimeGT interpolates between Archive events, if necessary, to find the times when the point crossed the given value.

Examples

```
TimeGT('tag1', 't', '*',40.0)
TimeGT('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeGT('digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeEq, TimeGE, TimeLE, TimeLT, TimeNE

TimeLE

Find the total time, within a range, when a point is less than or equal to a given value.

Format

```
TimeLE(tagname, starttime, endtime, value)
```

Arguments

`tagname`

A tagname.

`starttime`

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

`endtime`

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

`value`

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was less than or equal to the given value.

Exceptions

None.

Usage Note

TimeLE interpolates between Archive events, if necessary, to find the times when the point crossed the given value.

Examples

```
TimeLE('tag1', 't', '*',40.0)
TimeLE('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeLE('digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeEq, TimeGE, TimeGT, TimeLT, TimeNE

TimeLT

Find the total time, within a range, when a point is less than a given value.

Format

```
TimeLT(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
value
```

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was less than the given value.

Exceptions

None.

Usage Note

TimeLT interpolates between Archive events, if necessary, to find the times when the point crossed the given value.

Examples

```
TimeLT('tag1', 't', '*',40.0)
TimeLT('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeLT'digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeEq, TimeGE, TimeGT, TimeLE, TimeNE

TimeNE

Find the total time, within a range, when a point is unequal to a given value.

Format

```
TimeNE(tagname, starttime, endtime, value)
```

Arguments

```
tagname
```

A tagname.

```
starttime
```

Beginning of the time range to search. Relative times are relative to endtime, if endtime is not itself a relative time.

```
endtime
```

End of the time range to search. Relative times are relative to starttime, if starttime is not itself a relative time. This time must be after starttime.

```
value
```

Must be an integer or real number or digital state (character string); the value to search for.

Returns

The time period within the given range, for which the point was unequal to the given value.

Exceptions

None.

Examples

```
TimeNE('tag1', 't', '*',40.0)
TimeNE('digitaltag', '-1d', '*',TagVal('digitaltag', '14-Dec-97'))
TimeNE('digitaltag', '14-Dec-97', '*', "On")
```

See Also

TimeEq, TimeGE, TimeGT, TimeLE, TimeLT

Total

Return the sum of two or more arguments.

Format

```
Total(x1, x2, ..., xn)
```

Arguments

```
x1...xn
```

May be numbers or time periods, but all must be the same.

Returns

The total of the arguments. The result has the same data type as the arguments.

Exceptions

Arguments whose run-time values are digital states are not included in the total. If all values are digital states, **Total** returns an error value.

Examples

```
Total('tag1', 'tag2', TagVal('tag1', 'y'),40.0)
Total('t'-'y', '+1h')
```

Trim

Trim blanks on both sides of a string.

Format

```
Trim(str)
```

Arguments

```
str
```

A string.

Returns

The source string with leading and trailing blanks removed.

Exceptions

If **str** is not a string, an error value is returned.

Examples

```
Trim(" Stringtag ") = "Stringtag"
Trim(" Stringtag is a string tag.  ") = "Stringtag is a string tag."
```

See Also

LTrim, RTrim

Trunc

Truncate a number or time to the next lower unit.

Format

```
Trunc(x [, unit])
```

Arguments

x

Must be an integer or real number, time expression, or time period.

`unit (optional)`

The size of the unit to truncate to. If **x** is a number, unit must be a number. If **x** is a time expression or time period, unit must be a time period. If unit is omitted, **Trunc** truncates to the next lower integer (for a number) or second (for a time period).

Returns

The largest value smaller than **x** which is an integer multiple of unit. Returns the same data type as **x.** For more information, see the examples below.

Exceptions

If **x** is a string, or if unit is of the wrong data type, returns an error value.

Examples

Expression	Value	Comments
Trunc(12.999)	12.0	Truncate to next lower integer
Trunc(18.75, 10)	10.0	Truncate to next lower ten
Trunc('14-Dec-97 11:47','+1h')	14-Dec-97 11:00	Truncate to next lower hour
Trunc('18:47' – '15:00','+1h')	10800	Truncate period to next lower hour (returns period in seconds)

Note: *Trunc* to the next lower day results in a timestamp of the next lower day in UTC time, not local time.

Usage Note

If **x** is a time, and unit is omitted, this routine has no effect, as times are only accurate to one second.

See Also

Round

UCase

Convert a string to an uppercase string.

Format

```
UCase(strexp)
```

Arguments

```
strexp
```

Must be a string value.

Returns

An uppercase string.

Exceptions

If the argument is not a string, returns an error value.

Examples

```
UCase("Stringtag") = "STRINGTAG"
UCase('Stringtag') = "ERROR" if the Snapshot value for the stringtag equals
"Error"
```

See Also

LCase

Weekday

Extract the day of the week from a timestamp.

Format

```
Weekday(time)
```

Arguments

```
time
```

A time expression.

Returns

The day of the week of time, in the range 1–7, where 1 represents Sunday.

Exceptions

None.

Examples

```
Weekday('*')
Weekday('t')
```

See Also

Day, DaySec, Hour, Minute, Month, Second, Year, Yearday

Year

Extract the year from a time expression.

Format

```
Year(time)
```

Arguments

```
time
```

A time expression.

Returns

The year of time, in the range 1970–present.

Exceptions

None.

Examples

```
Year('*')
Year('t')
```

See Also

Day, DaySec, Hour, Minute, Month, Second, Weekday, Yearday

Yearday

Extract the day of the year from a time expression. The day of the year (also known as a Julian day) is an integer ranging from 1 to 366, where 1 represents January 1.

Format

```
Yearday(time)
```

Arguments

```
time
```

A time expression.

Returns

The day of the year of time, in the range 1–366, where 1 represents January 1.

Exceptions

None.

Examples

```
Yearday('*')
Yearday('t')
```

See Also

Day, DaySec, Hour, Minute, Month, Second, Weekday, Year

Chapter 5. PI STEAM FUNCTIONS REFERENCE

The **PI Steam Functions** module is an extension to the PI Performance Equations Scheduler. Steam Functions provide a complete set of functions for deriving the thermodynamic properties of steam and water within Performance Equations. PI Steam Functions support both English and SI units, and are based on the ASME (American Society of Mechanical Engineers) *Steam Tables*, *6th Ed*.

This chapter provides a comprehensive reference for setting up Steam calculations, and includes the following sections:

5.1 Steam Functions Overview

The PI PE Steam Functions Module (**PI Steam**) makes available functions that calculate the thermodynamic properties of steam within Performance Equations. The steam functions are also available in a COM library, *PISteamFunctions.dll*, which is distributed by other OSIsoft products, such as ACE.

For each steam function, there are two function calls: one in English units and another in *SI units*. The engineering units for the variables in the steam functions are listed in Table 5–1. Table 5–2 lists the supported functions in PI Steam.

Table 5–1. Engineering Units

Variable	English Unit	SI Unit	Conversion Factor (Eng to SI)
Temperature	degree F	degree C	(T - 32) / 1.8
Pressure	psia	kpa	6.894757
Volume	ft3/lbm	cc/g	62.42796
Enthalpy	BTU/lbm	J/g	2.326
Entropy	BTU/lbm/R	J/g/K	4.1868

Table 5–2. Supported Functions

Name	Function Description
StmEng_tsatp	Saturation temperature as a function of pressure.
StmEng_hsatp	Saturation enthalpy as a function of pressure.
StmEng_ssatp	Saturation entropy as a function of pressure.
StmEng_vsatp	Saturation vapor specific volume as a function of pressure.
StmEng_psatt	Saturation pressure as a function of temperature.
StmEng_hsatt	Saturation enthalpy as a function of temperature.
StmEng_ssatt	Saturation entropy as a function of temperature.
StmEng_vsatt	Saturation vapor specific volume as a function of temperature.
StmEng_vpt	Vapor specific volume as a function of pressure and temperature. (For saturated and super heated steam.)
StmEng_vptl	Water specific volume as a function of pressure and temperature.
StmEng_vph	Vapor specific volume as a function of pressure and enthalpy. (For wet and dry steam.)
StmEng_vps	Vapor specific volume as a function of pressure and entropy. (For wet and dry steam.)
StmEng_hpt	Enthalpy as a function of pressure and temperature. (For saturated and super heated steam.)
StmEng_hptl	Liquid enthalpy as a function of pressure and temperature. (For water.)
StmEng_hps	Enthalpy as a function of pressure and entropy. (For wet and dry steam.)
StmEng_spt	Entropy as a function of pressure and temperature. (For saturated and super heated steam.)
StmEng_sptl	Liquid entropy as a function of pressure and temperature. (For water.)
StmEng_sph	Entropy as a function of pressure and enthalpy. (For wet and dry steam.)
StmEng_tph	Temperature as a function of enthalpy and pressure. (For wet and dry steam.)
StmEng_tps	Temperature as a function of entropy and pressure. (For wet and dry steam.)
StmEng_xph	Steam quality (vapor fraction) as a function of enthalpy and pressure. (For wet steam.)
StmEng_xps	Steam quality (vapor fraction) as a function of entropy and pressure. (For wet steam.)
StmEng_hpx	Enthalpy as a function of pressure and steam quality. (For wet steam.)
StmEng_spx	Entropy as a function of pressure and steam quality. (For wet steam.)

5.1.1 Steam Functions Naming Convention

The formulas have the same naming convention as those callable by user program, i.e. STMENG_XXX for English units and STMSI_XXX for SI units. The formulas return the steam properties values and accept real number as arguments (i.e., argument type **R**). In case of error, the formulas return the digital states as shown in Table 5–3.

Table 5–3. Digital States Returned

Digital State	Description
INP OUTRANGE	Input condition out of computation range
NOT CONVERGE	Calculation failed to converge in iterative loop.

These digital states are standard and are installed with PI Server. The format for each formula is listed in the reference section.

5.2 Range of Steam Functions

Table 5–4 lists the valid range in English units of the input variables for each of the steam functions. The PI Steam functions should compute the same results as the equations given in the *ASME Steam Tables*, *6th Edition*. Besides the accuracy quoted by the *ASME Steam Tables*, you should be aware of the issues raised in *Functions that use Temperature and Pressure as Independent* Variables on page 193, and *Functions that use Enthalpy or Entropy as an Independent* Variable on page 194.

5.2.1 Functions that use Temperature and Pressure as Independent Variables

For the **HPT**, **SPT** and **VPT** functions, the steam has to be superheated or saturated dry. If the temperature and pressure are on the saturated curve, the calculated entropy, enthalpy or volume is the property of saturated vapor. If the input temperature is less than the saturated temperature for the input pressure, an *Input Out of Range* error is returned (i.e., digital state **INPOUTRANGE** for PI formulas or error code **-1** for user-callable functions). Since measurements are not exact, these PT functions can tolerate an error margin:

```
if (0 > Tsat - Tin > error margin) then saturated steam
```

The default error margin is 0.5 degree F.

Table 5–4. Input Range for Each Function

Function	Temp (deg f)	Pressure (psia)	Enthalpy (btu/lb)	Entropy (btu/lb/r)	Quality
StmEng_tsatp		0.088589 to 3208.2			
StmEng_hsatp		0.088589 to 3208.2			
StmEng_ssatp		0.088589 to 3208.2			
StmEng_vsatp		0.088589 to 3208.2			
StmEng_psatt	32. to 705.47				
StmEng_hsatt	32. to 705.47				
StmEng_ssatt	32. to 705.47				

Function	Temp (deg f)	Pressure (psia)	Enthalpy (btu/lb)	Entropy (btu/lb/r)	Quality
StmEng_vsatt	32. to 705.47				
StmEng_vpt, StmEng_vptl	32. to 1600	0.088589 to 16000.			
StmEng_vph		0.088589 to 16000.	-1 to 1860.		
StmEng_vps		0.088589 to 16000.		-0.1 to 3.0	
StmEng_hpt, StmEng_hptl	32. to 1600	0.088589 to 16000.			
StmEng_hps		0.088589 to 16000.		-0.1 to 3.0	
StmEng_spt, StmEng_sptl	32. to 1600	0.088589 to 16000.			
StmEng_sph		0.088589 to 16000.	-1 to 1860.		
StmEng_tph		0.088589 to 16000.	-1 to 1860.		
StmEng_tps		0.088589 to 16000.		-0.1 to 3.0	
StmEng_xph		0.088589 to 3208.2	-1 to 1860.		
StmEng_xps		0.088589 to 3208.2		-0.1 to 3.0	
StmEng_hpx		0.088589 to 3208.2			0 to 1
StmEng_spx		0.088589 to 3208.2			0 to 1

5.2.2 Functions that use Enthalpy or Entropy as an Independent Variable

For the **VPH**, **VPS**, **HPS**, **SPH**, **TPH** and **TPS** functions, the valid ranges cover both the superheated and the wet steam. However, even though the ASME listed valid range for enthalpy goes from -1 to 1860 BTU/lbm and the entropy ranges from -0.1 to 3.0 BTU/lbm/degR, there are some combination of pressure and enthalpy or entropy that correspond to compressed water rather than steam. Hence, these input conditions will generate an error state.

For the wet steam region, the ASME routines use the Clapeyron equation and the saturated vapor values to compute the result. For the **VPH** and **VPS** functions, the computed volume can differ by a few percents as compared to the volume calculated from the saturated vapor volume, saturated liquid volume and the vapor fraction. The difference in the computed volume increases as the moisture content of the vapor mixture increases. However, since the practical use of the steam function involves steam with vapor fraction higher than 0.5, the ASME equation is not modified.

For the **HPS** and **SPH** functions in the wet steam region, the ASME routines use the equation ($H_{vap} = T*S_{vap}$) and the saturated vapor values to compute the result. The functions do not check if the input enthalpy or entropy is less than those of the saturated liquid and the extrapolated value is returned.

For the **TPH** and **TPS** function in the wet steam region, the ASME routines just check whether the input enthalpy or entropy is less than that of the saturated vapor at the given pressure. If the input enthalpy or entropy is less than the saturated vapor value, the saturated temperature is returned as the answer. The functions do not consider the case where the input enthalpy or input entropy is less than that of the saturated liquid an error state.

Similarly, for the **XPH** and **XPS** functions, the ASME routines do not consider the input enthalpy or entropy out of bound even when they are greater than saturated vapor properties or less than saturated liquid properties. The functions return 1.0 as the vapor fraction when the input enthalpy or entropy is greater than the saturated vapor properties. Input enthalpy or entropy less than the saturated liquid properties results in negative vapor fraction rather than an error state.

Though the ASME routines have inadequate checks for the wet steam region, OSI did not modify these routines because in reality, the steam functions are not used in regions where the input checks would cause a problem.

One final feature, for pressure above the critical point (3208.2 psia), the **VPH**, **VPS**, **HPS**, **SPH**, **TPH** and **TPS** functions compute valid results for states corresponded to temperature greater than 682 degrees F. but less than the critical temperature, 705.47 degrees F, even though these states are considered compressed water rather than steam.

5.3 Steam Property Reference States

The ASME establishes the following reference states:

Triple Point

Triple point of water is at 273.16 degree K or 0.01 degree C or 0.018 degree F.

Celsius Scale

The Celsius temperature is exactly T_k - 273.15.

Critical Point

Critical point of steam is at 647.3 degree K and 22,120 kpa, or 705.47 degree F and 3208.3 psia.

Reference State

The specific internal energy and specific entropy of the liquid phase were fixed at zero at the triple point of water.

5.4 Steam Functions Reference

This section provides a detailed reference of Steam Functions.

StmEng_TsatP

Calculates the saturation temperature as a function of pressure—all variables expressed in English units.

Format

```
StmEng_TsatP(P)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

Returns

Computed saturation temperature in degrees F. or *Error* digital state.

Sample Values

Pressure	**Temperature**
10.	193.21
100.	327.82
1000.	544.58

StmEng_HsatP

Calculates the saturated vapor specific enthalpy as a function of pressure—all variables expressed in English units.

Format

```
StmEng_HsatP(P)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

Returns

Computed specific enthalpy for saturated vapor in BTU/lbm or *Error* digital state.

Sample Values

Pressure	**Vapor Enthalpy**
10.	1143.4
100.	1187.2
1000.	1192.9

StmEng_SsatP

Calculates the saturated vapor specific entropy as a function of pressure—all variables expressed in English units.

Format

```
StmEng_SsatP(P)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

Returns

Computed saturated vapor specific entropy in BTU/lbm/R or *Error* digital state.

Sample Values

Pressure	**Entropy**
10.	1.7879
100.	1.6027
1000.	1.3910

StmEng_VsatP

Calculates the saturated vapor specific volume as a function of pressure—all variables expressed in English units.

Format

```
StmEng_VsatP(P)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

Returns

Saturated vapor specific volume in ft^3/lbm or *Error* digital state.

Sample Values

Pressure	**Volume**
10.	38.42
100.	4.431
1000.	0.44596

StmEng_PsatT

Calculates the saturation pressure as a function of temperature—all variables expressed in English units.

Format

```
StmEng_PsatT(T)
```

Arguments

```
T
```

Steam temperature in degree F. The valid range is 32 to 705.47 degree F.

Returns

Computed saturation pressure of the steam in psia or *Error* digital state.

Sample Values

Temperature	**Pressure**
100.	0.9492
400.	247.26
700.	3094.33

StmEng_HsatT

Calculates the saturated vapor specific enthalpy as a function of temperature—all variables expressed in English units.

Format

```
StmEng_HsatT(T)
```

Arguments

```
T
```

Steam temperature in degree F. The valid range is 32 to 705.47 degree F.

Returns

Computed specific enthalpy for saturated vapor in BTU/lbm or *Error* digital state.

Sample Values

Temperature	Vapor Enthalpy
100.	1105.1
400.	1201.0
700.	995.2

StmEng_SsatT

Calculates the saturated vapor specific entropy as a function of temperature—all variables expressed in English units.

Format

```
StmEng_SsatT(T)
```

Arguments

`T`

Steam temperature in degree F. The valid range is 32 to 705.47 degree F.

Returns

Computed saturated vapor specific entropy in BTU/lbm/R or- *Error* digital state.

Sample Values

Temperature	**Entropy**
100.	1.9825
400.	1.52735
700.	1.1390

StmEng_VsatT

Calculates the saturated vapor specific volume as a function of temperature—all variables expressed in English units.

Format

```
StmEng_VsatT(T)
```

Arguments

```
T
```

Steam temperature in degree F. The valid range is 32 to 705.47 degree F

Returns

Computed saturated vapor specific volume in ft^3/lbm or *Error* digital state

Sample Values

Temperature	**Volume**
100.	350.39
400.	1.863
700.	0.0752

StmEng_VPT

Calculates the vapor specific volume as a function of pressure and temperature—all variables expressed in English units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_VPT(P, T)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`T`

Steam temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed vapor specific volume in ft^3/lbm or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Volume**
300.	1000.	2.8585
800.	1000.	1.047
1400.	1000.	0.5809
5000.	1000.	0.13118

StmEng_VPTL

Calculates the liquid specific volume as a function of pressure and temperature—all variables expressed in English units. Only use for liquid water condition. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_VPTL(P, T)
```

Arguments

```
P
```

Pressure of the water in psia. The valid range is 0.088589 to 16000 psia.

```
T
```

Water temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed liquid specific volume in ft^3/lbm or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Volume**
300.	100.	0.016115
800.	100.	0.016091
1400.	100.	0.016062
5000.	100.	0.015897

StmEng_VPH

Calculates the vapor specific volume as a function of pressure and enthalpy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_VPH(P, H)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`H`

Specific enthalpy of the steam in BTU/lbm. The valid range is -1 to 1860 BTU/lbm.

Returns

Computed vapor specific volume in ft^3/lbm or *Error* digital state.

Sample Values

Pressure	**Enthalpy**	**Volume**	**State**
300.	1526.2	2.8585	Superheated
800.	1511.4	1.047	Superheated
1400.	1493.2	0.5809	Superheated
5000.	1364.6	0.13118	Superheated
300.	1122.	1.3904	90 % vapor

Note: The computed result may differ slightly from that computed using the saturated liquid volume, saturated vapor volume and vapor fraction. The difference increases with the moisture content of the steam.

StmEng_VPS

Calculates the vapor specific volume as a function of pressure and entropy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_VPS(P, S)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`S`

Specific entropy of the steam in BTU/lbm/R. The valid range is -0.1 to 3.0 BTU/lbm/R.

Returns

Computed vapor specific volume in ft^3/lbm or *Error* digital state.

Sample Values

Pressure	Entropy	Volume	State
300.	1.7964	2.8585	Superheated
800.	1.6807	1.047	Superheated
1400.	1.6096	0.5809	Superheated
5000.	1.4001	0.13118	Superheated
300.	1.4183	1.3904	90 % vapor

Note: The computed result may differ slightly from that computed using the saturated liquid volume, saturated vapor volume and vapor fraction. The difference increases with the moisture content of the steam.

StmEng_HPT

Calculates the vapor specific enthalpy as a function of pressure and temperature—all variables expressed in English units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_HPT(P, T)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`T`

Steam temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed vapor specific enthalpy in BTU/lbm or *Error* digital state.

Sample Values

Pressure	Temperature	Enthalpy
300.	1000.	1526.2
800.	1000.	1511.4
1400.	1000.	1493.2
5000.	1000.	1364.6

StmEng_HPTL

Calculates the liquid specific enthalpy as a function of pressure and temperature—all variables expressed in English units. Only use for compressed water. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_HPTL(P, T)
```

Arguments

`P`

Pressure of the water in psia. The valid range is 0.088589 to 16000 psia.

`T`

Water temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed liquid specific enthalpy in BTU/lbm or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Enthalpy**
300.	100.	68.788
800.	100.	70.106
1400.	100.	71.684
5000.	100.	81.081

StmEng_HPS

Calculates the vapor specific enthalpy as a function of pressure and entropy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_HPS(P, S)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

```
S
```

Specific entropy of the steam in BTU/lbm/R. The valid range is -0.1 to 3.0 BTU/lbm/R.

Returns

Computed vapor specific enthalpy in BTU/lbm or *Error* digital state.

Sample Values

Pressure	Entropy	Enthalpy	State
300.	1.7964	1526.2	Superheated
800.	1.6807	1511.4	Superheated
1400.	1.6096	1493.2	Superheated
5000.	1.4001	1364.6	Superheated
300.	1.4183	1122.	90 % vapor

Note: Even if the input entropy is less than that of the saturated liquid, the function still computes the enthalpy by extrapolation without setting an error state.

StmEng_SPT

Calculates the vapor specific entropy as a function of pressure and temperature—all variables expressed in English units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_SPT(P, T)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

```
T
```

Steam temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed vapor specific entropy in BTU/lbm/R or- *Error* digital state.

Sample Values

Pressure	**Temperature**	**Entropy**
300.	1000.	1.7964
800.	1000.	1.6807
1400.	1000.	1.6096
5000.	1000.	1.4001

StmEng_SPTL

Calculates the liquid specific entropy as a function of pressure and temperature—all variables expressed in English units. Only use for compressed water. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmEng_SPTL(P, T)
```

Arguments

```
P
```

Pressure of the water in psia. The valid range is 0.088589 to 16000 psia.

```
T
```

Water temperature in degree F. The valid range is 32 to 1600 degree F.

Returns

Computed liquid specific entropy in BTU/lbm/R or- *Error* digital state.

Sample Values

Pressure	**Temperature**	**Entropy**
300.	100.	0.12936
800.	100.	0.12905
1400.	100.	0.12868
5000.	100.	0.12645

StmEng_SPH

Calculates the vapor specific entropy as a function of pressure and enthalpy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_SPH(P, H)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`H`

Computed vapor specific enthalpy in BTU/lbm. The valid range is -1.0 to 1860 BTU/lbm.

Returns

Computed vapor specific entropy in BTU/lbm/R or- *Error* digital state.

Sample Values

Pressure	Enthalpy	Entropy	State
300.	1526.2	1.7964	Superheated
800.	1511.4	1.6807	Superheated
1400.	1493.2	1.6096	Superheated
5000.	1364.6	1.4001	Superheated
300.	1122.	1.4183	90 % vapor

Note: Even if the input enthalpy is less than that of the saturated liquid, the function still computes the entropy by extrapolation without setting an error state.

StmEng_TPH

Calculates the steam temperature as a function of pressure and enthalpy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_TPH(P, H)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

`H`

Specific enthalpy of the steam in BTU/lbm. The valid range is -1 to 1860 BTU/lbm.

Returns

Computed steam temperature in degree F. or *Error* digital state.

Sample Values

Pressure	**Enthalpy**	**Temperature**	**State**
300.	1526.2	1000.	Superheated
800.	1511.4	1000.	Superheated
1400.	1493.2	1000.	Superheated
5000.	1364.6	1000.	Superheated
300.	1122.	417.35	90 % vapor

Note: Even if the input enthalpy is less than that of the saturated liquid, the function still returns the saturated temperature as the answer without setting an error state.

StmEng_TPS

Calculates the steam temperature as a function of pressure and entropy—all variables expressed in English units. Use for both superheated or wet steam.

Format

```
StmEng_TPS(P, S)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 16000 psia.

```
S
```

Specific entropy of the steam in BTU/lbm/R. The valid range is -0.1 to 3.0 BTU/lbm/R.

Returns

Computed steam temperature in degree F. or *Error* digital state.

Sample Values

Pressure	Entropy	Temperature	State
300.	1.7964	1000.	Superheated
800.	1.6807	1000.	Superheated
1400.	1.6096	1000.	Superheated
5000.	1.4001	1000.	Superheated
300.	1.4183	417.35	90 % vapor

Note: Even if the input entropy is less than that of the saturated liquid, the function still returns the saturated temperature as the answer without setting an error state.

StmEng_XPH

Calculates the steam quality (vapor fraction) as a function of pressure and enthalpy—all variables expressed in English units. Use only for wet steam.

Format

```
StmEng_XPH(P, H)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

`H`

Specific enthalpy of the steam in BTU/lbm. The valid range is -1 to 1860 BTU/lbm.

Returns

Computed steam quality (vapor fraction) or *Error* digital state.

Sample Values

Pressure	**Enthalpy**	**Steam Quality**
300.	1122.0	0.9
800.	1130.4	0.9
1400.	1117.7	0.9

Note: If the input enthalpy is greater than that of the saturated vapor, the function returns 1.0 as vapor fraction. If the input enthalpy is less than that of the saturated liquid, the function would compute negative vapor fraction without setting an error state.

StmEng_XPS

Calculates the steam quality (vapor fraction) as a function of pressure and entropy—all variables expressed in English units. Use only for wet steam.

Format

```
StmEng_XPS(P, S)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

`S`

Specific entropy of the steam in BTU/lbm/R. The valid range is -0.1 to 3.0 BTU/lbm/R or *Error* digital state.

Returns

Computed steam quality (vapor fraction).

Sample Values

Pressure	Entropy	Steam Quality
300.	1.4183	0.9
800.	1.3458	0.9
1400.	1.2923	0.9

Note: If the input entropy is greater than that of the saturated vapor, the function returns 1.0 as vapor fraction. If the input entropy is less than that of the saturated liquid, the function would compute negative vapor fraction without setting an error state.

StmEng_HPX

Calculates the steam specific enthalpy as a function of pressure and quality (vapor fraction)—all variables expressed in English units. Use only for wet steam.

Format

```
StmEng_HPX(P, X)
```

Arguments

`P`

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

`X`

Steam quality (vapor fraction). Valid range is from 0.0 to 1.0.

Returns

Computed specific enthalpy of the steam in BTU/lbm or *Error* digital state.

Sample Values

Pressure	**Steam Quality**	**Enthalpy**
300.	0.9	1122.0
800.	0.9	1130.4
1400.	0.9	1117.7

StmEng_SPX

Calculates the steam specific entropy as a function of pressure and quality (vapor fraction)—all variables expressed in English units. Use only for wet steam.

Format

```
StmEng_SPX(P, X)
```

Arguments

```
P
```

Pressure of the steam in psia. The valid range is 0.088589 to 3208.2 psia.

```
X
```

Steam quality (vapor fraction). Valid range is from 0.0 to 1.0.

Returns

Computed specific entropy of the steam in BTU/lbm/R or- *Error* digital state.

Sample Values

Pressure	Steam Quality	Entropy
300.	0.9	1.4183
800.	0.9	1.3458
1400.	0.9	1.2923

StmSI_TsatP

Calculates the saturation temperature as a function of pressure—all variables expressed in SI units.

Format

```
StmSI_TsatP(P)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

Returns

Computed saturation temperature in degree C. or *Error* digital state.

Sample Values

Pressure	**Temperature**
50.	81.345
1000.	179.88
10000.	310.96

StmSI_HsatP

Calculates the saturated vapor specific enthalpy as a function of pressure—all variables expressed in SI units.

Format

```
StmSI_HsatP(P)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

Returns

Computed specific enthalpy for saturated vapor in J/g.

Sample Values

Pressure	**Vapor Enthalpy**
50.	2646.0
1000.	2776.2
10000.	2727.7

StmSI_SsatP

Calculates the saturated vapor specific entropy as a function of pressure—all variables expressed in SI units.

Format

```
StmSI_SsatP(P)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

Returns

Computed saturated vapor specific entropy in J/g/K or *Error* digital state.

Sample Values

Pressure	**Entropy**
50.	7.5947
1000.	6.5828
10000.	5.6198

StmSI_VsatP

Calculates the saturated vapor specific volume as a function of pressure—all variables expressed in SI units.

Format

```
StmSI_VsatP(P)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

Returns

Computed saturated vapor specific volume in cc/g or *Error* digital state.

Sample Values

Pressure	**Volume**
50.	3240.2
1000.	194.29
10000.	18.041

StmSI_PsatT

Calculates the saturation pressure as a function of temperature—all variables expressed in SI units.

Format

```
StmSI_PsatT(T)
```

Arguments

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 374.15 degree C.

Returns

Computed saturation pressure of the steam in kpa or Error digital state.

Sample Values

Temperature	**Pressure**
50.	12.335
200.	1554.9
350.	16535.

StmSI_HsatT

Calculates the saturated vapor specific enthalpy as a function of temperature—all variables expressed in SI units.

Format

```
StmSI_HsatT(T)
```

Arguments

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 374.15 degree C.

Returns

Computed specific enthalpy for saturated vapor in J/g or *Error* digital state.

Sample Values

Temperature	**Vapor Enthalpy**
50.	2592.2
200.	2790.9
350.	2567.7

StmSI_SsatT

Calculates the saturated vapor specific entropy as a function of temperature—all variables expressed in SI units.

Format

```
StmSI_SsatT(T)
```

Arguments

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 374.15 degree C.

Returns

Computed saturated vapor specific entropy in J/g/K or *Error* digital state.

Sample Values

Temperature	**Entropy**
50.	8.0776
200.	6.4278
350.	5.2177

StmSI_VsatT

Calculates the saturated vapor specific volume as a function of temperature—all variables expressed in SI units.

Format

```
StmSI_VsatT(T)
```

Arguments

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 374.15 degree C

Returns

Computed saturated vapor specific volume in cc/g or *Error* digital state.

Sample Values

Temperature	Volume
50.	12046.
200.	127.16
350.	8.7991

StmSI_VPT

Calculates the vapor specific volume as a function of pressure and temperature—all variables expressed in SI units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_VPT(P, T)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed vapor specific volume in cc/g or *Error* digital state.

Sample Values

Pressure	Temperature	Volume
2500.	600.	159.21
5000.	600.	78.616
10000.	600.	38.320
40000.	600.	8.0884

StmSI_VPTL

Calculates the liquid specific volume as a function of pressure and temperature—all variables expressed in SI units. Only use for compressed water. An error of **-1**(or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_VPTL(P, T)
```

Arguments

```
P
```

Pressure of the water in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Water temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed liquid specific volume in cc/g or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Volume**
2500.	100.	1.04245
5000.	100.	1.04116
10000.	100.	1.03861
40000.	100.	1.02438

StmSI_VPH

Calculates the vapor specific volume as a function of pressure and enthalpy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_VPH(P, H)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
H
```

Specific enthalpy of the steam in J/g. The valid range is -2.326 to 4326.36 J/g.

Returns

Computed vapor specific volume in cc/g or *Error* digital state.

Sample Values

Pressure	Enthalpy	Volume	State
2500.	3685.1	159.21	Superheated
5000.	3664.5	78.616	Superheated
10000.	3622.7	38.320	Superheated
40000.	3346.4	8.0884	Superheated
2500.	2617.0	72.04	90 % vapor

Note: The computed result may differ slightly from that computed using the saturated liquid volume, saturated vapor volume and vapor fraction. The difference increases with the moisture content of the steam.

StmSI_VPS

Calculates the vapor specific volume as a function of pressure and entropy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_VPS(P, S)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
S
```

Specific entropy of the steam in J/g/K. The valid range is -0.41868 to 12.5604 J/g/K.

Returns

Computed vapor specific volume in cc/g or *Error* digital state.

Sample Values

Pressure	**Entropy**	**Volume**	**State**
2500.	7.5956	159.21	Superheated
5000.	7.2578	78.616	Superheated
10000.	6.9013	38.320	Superheated
40000.	6.0135	8.0884	Superheated
2500.	5.8837	72.04	90 % vapor

Note: The computed result may differ slightly from that computed using the saturated liquid volume, saturated vapor volume and vapor fraction. The difference increases with the moisture content of the steam.

StmSI_HPT

Calculates the vapor specific enthalpy as a function of pressure and temperature—all variables expressed in SI units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_HPT(P, T)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed vapor specific enthalpy in J/g or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Enthalpy**
2500.	600.	3685.1
5000.	600.	3664.5
10000.	600.	3622.7
40000.	600.	3346.4

StmSI_HPTL

Calculates the liquid specific enthalpy as a function of pressure and temperature—all variables expressed in SI units. Only use for compressed water. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_HPTL(P, T)
```

Arguments

```
P
```

Pressure of the water in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Water temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed liquid specific enthalpy in J/g or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Enthalpy**
2500.	100.	420.86
5000.	100.	422.74
10000.	100.	426.50
40000.	100.	449.22

StmSI_HPS

Calculates the vapor specific enthalpy as a function of pressure and entropy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_HPS(P, S)
```

Arguments

`P`

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

`S`

Specific entropy of the steam in J/g/K. The valid range is -0.41868 to 12.5604 J/g/K.

Returns

Computed vapor specific enthalpy in J/g or *Error* digital state.

Sample Values

Pressure	**Entropy**	**Enthalpy**	**State**
2500.	7.5956	3685.1	Superheated
5000.	7.2578	3664.5	Superheated
10000.	6.9013	3622.7	Superheated
40000.	6.0135	3346.4	Superheated
2500.	5.8837	2617.0	90 % vapor

Note: Even if the input entropy is less than that of the saturated liquid, the function still computes the enthalpy by extrapolation without setting an error state.

StmSI_SPT

Calculates the vapor specific entropy as a function of pressure and temperature—all variables expressed in SI units. Only use for superheated or dry saturated steam. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature lower than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_SPT(P, T)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Steam temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed vapor specific entropy in J/g/K or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Entropy**
2500.	600.	7.5956
5000.	600.	7.2578
10000.	600.	6.9013
40000.	600.	6.0135

StmSI_SPTL

Calculates the liquid specific entropy as a function of pressure and temperature—all variables expressed in SI units. Only use for compressed water. An error of **-1** (or *input out of range* digital state) will be resulted for input temperature higher than the saturation temperature for the input pressure. However, the function computes for the full range of temperature for input pressure greater than the critical pressure.

Format

```
StmSI_SPTL(P, T)
```

Arguments

```
P
```

Pressure of the water in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
T
```

Water temperature in degree C. The valid range is 0.0 to 871.11 degree C.

Returns

Computed liquid specific entropy in J/g/K or *Error* digital state.

Sample Values

Pressure	**Temperature**	**Entropy**
2500.	100.	1.305
5000.	100.	1.30304
10000.	100.	1.29919
40000.	100.	1.27714

StmSI_SPH

Calculates the vapor specific entropy as a function of pressure and enthalpy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_SPH(P, H)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
H
```

Computed vapor specific enthalpy in J/g. The valid range is -2.326 to 4326.36 J/g.

Returns

Computed vapor specific entropy in J/g/K or *Error* digital state.

Sample Values

Pressure	Enthalpy	Entropy	State
2500.	3685.1	7.5956	Superheated
5000.	3664.5	7.2578	Superheated
10000.	3622.7	6.9013	Superheated
40000.	3346.4	6.0135	Superheated
2500.	2617.0	5.8837	90 % vapor

Note: Even if the input enthalpy is less than that of the saturated liquid, the function still computes the entropy by extrapolation without setting an error state.

StmSI_TPH

Calculates the steam temperature as a function of pressure and enthalpy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_TPH(P, H)
```

Arguments

`P`

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

`H`

Specific enthalpy of the steam in J/g. The valid range is -2.326 to 4326.36 J/g.

Returns

Computed steam temperature in degree C. or *Error* digital state.

Sample Values

Pressure	**Enthalpy**	**Temperature**	**State**
2500.	3685.1	600.	Superheated
5000.	3664.5	600.	Superheated
10000.	3622.7	600.	Superheated
40000.	3346.4	600.	Superheated
2500.	2617.0	223.94	90 % vapor

Note: Even if the input enthalpy is less than that of the saturated liquid, the function still returns the saturated temperature as the answer without setting an error state.

StmSI_TPS

Calculates the steam temperature as a function of pressure and entropy—all variables expressed in SI units. Use for both superheated or wet steam.

Format

```
StmSI_TPS(P, S)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 110316.0 kpa.

```
S
```

Specific entropy of the steam in J/g/K. The valid range is -0.41868 to 12.5604 J/g/K.

Returns

Computed steam temperature in degree C. or *Error* digital state.

Sample Values

Pressure	Entropy	Temperature	State
2500.	7.5956	600.	Superheated
5000.	7.2578	600.	Superheated
10000.	6.9013	600.	Superheated
40000.	6.0135	600.	Superheated
2500.	5.8837	223.94	90 % vapor

Note: Even if the input entropy is less than that of the saturated liquid, the function still returns the saturated temperature as the answer without setting an error state.

StmSI_XPH

Calculates the steam quality (vapor fraction) as a function of pressure and enthalpy—all variables expressed in SI units. Use only for wet steam.

Format

```
StmSI_XPH(P, H)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

```
H
```

Specific enthalpy of the steam in J/g. The valid range is -2.326 to 4326.36 J/g.

Returns

Computed steam quality (vapor fraction) or *Error* digital state.

Sample Values

Pressure	**Enthalpy**	**Steam Quality**
2500.	2617.0	0.9
5000.	2630.2	0.9
10000.	2595.8	0.9

Note: If the input enthalpy is greater than that of the saturated vapor, the function returns 1.0 as vapor fraction. If the input enthalpy is less than that of the saturated liquid, the function would compute negative vapor fraction without setting an error state.

StmSI_XPS

Calculates the steam quality (vapor fraction) as a function of pressure and entropy—all variables expressed in SI units. Use only for wet steam.

Format

```
StmSI_XPS(P, S)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

```
S
```

Specific entropy of the steam in J/g/K. The valid range is -0.41868 to 12.5604 J/g/K.

Returns

Computed steam quality (vapor fraction) or *Error* digital state.

Sample Values

Pressure	Entropy	Steam Quality
2500.	5.8837	0.9
5000.	5.6682	0.9
10000.	5.3939	0.9

Note: If the input entropy is greater than that of the saturated vapor, the function returns 1.0 as vapor fraction. If the input entropy is less than that of the saturated liquid, the function would compute negative vapor fraction without setting an error state.

StmSI_HPX

Calculates the steam specific enthalpy as a function of pressure and quality (vapor fraction)—all variables expressed in SI units. Use only for wet steam.

Format

```
StmSI_HPX(P, X)
```

Arguments

```
P
```

Pressure of the steam in *kpa*. The valid range is 0.6108 to 22119.8 kpa.

```
X
```

Steam quality (vapor fraction). Valid range is from 0.0 to 1.0.

Returns

Computed specific enthalpy of the steam in J/g or *Error* digital state.

Sample Values

Pressure	Steam Quality	Enthalpy
2500.	0.9	2617.0
5000.	0.9	2630.2
10000.	0.9	2595.8

StmSI_SPX

Calculates the steam specific entropy as a function of pressure and quality (vapor fraction)—all variables expressed in SI units. Use only for wet steam.

Format

```
StmSI_SPX(P, X)
```

Arguments

```
P
```

Pressure of the steam in kpa. The valid range is 0.6108 to 22119.8 kpa.

```
X
```

Steam quality (vapor fraction). Valid range is from 0.0 to 1.0.

Returns

Computed specific entropy of the steam in J/g/K or *Error* digital state.

Sample Values

Pressure	**Steam Quality**	**Entropy**
2500.	0.9	5.8837
5000.	0.9	5.6682
10000.	0.9	5.3939

Chapter 6. PI BATCH DATABASE

Most processes have repeatable time segments or stages. The **PI Batch Subsystem** maps process or manufacturing events to slices of time and data, and stores these batch- and process-based events hierarchically in the PI Data Archive as *Batches*, *Unit Batches*, or *Sub Batches*. This capability enables powerful data and process analysis for both traditional and non-traditional batch processes.

Many manufacturing companies find it useful to track their products in discrete batches rather than as a time-continuous product. PI Batch can define and record sequences such as events in discrete manufacturing, paper reels, steel coils, product or grade changes, turbine startups, and lab runs. While industries such as chemicals and pharmaceuticals use PI Batch to track and analyze batches, it is also widely used in non-batch applications to identify and track process events. Along with the use of audit trail support, PI Batch and the PI System become an integral component of a validated reporting environment in compliance with 21 CFR Part 11.

This chapter describes the functionality provided by the **PI Batch Subsystem**, from configuration to interaction with the resulting batch data, and includes the following topics:

6.1 PI Batch Overview

Traditional batch applications are common in industries like chemical, food and beverage, and pharmaceutical manufacturing. Batch processing has also been used in applications in which a sequence of steps occurs, such as burner startup and shutdown, to determine whether or not proper sequencing took place. Furthermore, batch processing has been used in applications that are not typically considered pure batch processes to correlate process data that has been generated during say an 8-hour shift or a 24-hour time period. Comparison of various parameters during such shifts or time periods often leads to valuable insight about the underlying process.

PI Batch is used in conjunction with its companion Client application, PI BatchView, which allows you to search, select, trend, and compare events that have been collected by PI Batch and stored in the PI System. Earlier versions of PI BatchView were based on the PI-API, and more recent versions are based on the PI-SDK.

Batch activity is indexed on the following parameters:

- Time
- Unit
- Batch ID
- Product ID

Once the batch activity is recorded, specific batches or groups of batches can be searched and retrieved. The batch search results may then be used to frame process data from the PI Archive in the context of the selected batches.

6.1.1 The PI Batch Subsystem (BSS), PI Batch Database (PBD), and PI Batch Generator (PIBaGen)

The **PI Batch Subsystem (BSS)** was introduced with PI Server 3.1. It provides batch processing functionality: configuration, monitoring, and query processing. It continues to be installed and supported, but it is no longer the only (or preferred) technology for batch processing.

❑ For information about the **PI Batch Subsystem,** read this chapter.

The **PI Batch Database (BDB)** was introduced with PI Server 3.3 and PI-SDK 1.1. It provides enhanced batch information processing support.

❑ For information about the **PI Batch Database,** refer to the ***PI-SDK User Manual.***

The **PI Batch Generator (PIBaGen) Interface** was introduced with the PI Batch Database. It is now included with PI Server Applications. It is a PI-SDK-based interface that reads process unit configuration from the PI Module Database and writes into the PI Batch Database. The interface monitors the PI Server for events that trigger the beginning of a batch and then stores information about each batch, such as Batch ID and Product ID, into the PI Data Archive.

❑ For information about the **PI Batch Generator**, refer to the ***PI Batch Generator (PIBaGen) User Manual***.

The **PI Batch Database** and the **PI Batch Generator** are independent of the **PI Batch Subsystem**. All three of these system components may be used in parallel.

❑ For a comparison of the **PI Batch Subsystem** and **PI Batch Database** refer to ***PI Batch Database Support of the PI Batch Subsystem.*** This document also explains how to access BSS batches from the BDB, and vice versa.

To search, select, trend, and compare batches of interest, client products such as PI-BatchView are available.

6.1.2 Compatibility of PI-API Batch Applications for PI2 (OpenVMS) Servers

PI-API batch applications developed against PI BA in a PI2 Server (OpenVMS) will still function correctly when run against a PI3 Server with the PI Batch Subsystem.

6.1.3 Glossary of Batch Terms

The following Batch terms are used in this chapter.

Term	Definition
Batch	A batch represents a span of time on a unit.
Unit	Defines the name of the equipment set on which the batch activity takes place. Unit definitions are not limited to a single piece of equipment. For example, a unit could be a single reactor or a group of reactors and related equipment.
BatchID	This is a name given to a batch of material. The BatchID is not required to be unique.
ProductID	Describes a specific material or class of materials. This term is used in batch applications that use equipment to produce a variety of different materials. A ProductID is not required for a batch definition, but it does provide a useful search index.

6.2 Installation

The PI Batch Subsystem is automatically installed on new installations or upgrades of the PI3 Server, but a valid license is required in order to use it. The *pibatch.exe* executable is located in *\PI\bin*. The installation itself will not result in the creation of any units or batches, but the installed **piconfig** script *\PI\adm\pibatch.dif* can be used to configure an example unit and enable the creation of sample batches. The *pibatch.dif* script is discussed in more detail in section 6.7.

The PI Batch Subsystem must started first, to configure any units or to create batches. Startup will typically occur automatically either at boot time or when the PI Server startup script is executed.

6.3 Configuration

All configuration for the PI Batch Subsystem must be performed with the command-line utility **piconfig**. See chapter 11, *The Piconfig Utility*, in the ***PI Server System Management Guide,*** for information about using the **piconfig** utility.

6.3.1 Unit Configuration

The starting point for all batch configurations is the unit, the piece or set of equipment where batches occur. There are two fundamental rules for units:

- ❑ The unit name must be unique.
- ❑ The unit may only process one batch at a time.

PI Batch Unit (PIBAUNIT) Table

The configuration for all units is stored in the data file *\PI\dat\pibaunit1.nt* and is accessed exclusively via the **piconfig** table named **PIBAUNIT**. The configuration for a unit involves several attributes of information, and the following list includes each attribute's name, data type, default value ("**D**:"), and current value ("**C**:"):

```
* (Ls - ) PIconfig> @table PIBAUNIT
* (Ls - PIBAUNIT) PIconfig> @?atr
1 - UnitName String D: C:
2 - NEWUnitName String D: C:
3 - ActiveTag String D: C:
4 - ActiveType String D: pulse C:
5 - BIDExpr String D: C:
6 - DataAccess String D: o:rw g:r w:r C:
7 - DataGroup String D: piadmin C:
8 - DataOwner String D: piadmin C:
9 - Description String D: C:
10 - EvalDelay String D: 0 C:
11 - MergeConsecutive String D: 0 C:
12 - ProdExpr String D: C:
13 - UnitAccess String D: o:rw g:r w:r C:
14 - UnitGroup String D: piadmin C:
15 - UnitOwner String D: piadmin C:
```

Table 6–1 provides a complete description of each of the unit attributes.

Table 6–1. PIBAUNIT Table Attributes

Attribute	**Description**			
UnitName	This defines the unique unit name, which is the primary index of the PIBAUNIT table. The name cannot include the **'\'** character but can be renamed.			
NEWUnitName	Used to rename an existing unit.			
ActiveTag	The PI tag that defines whether a batch is active or inactive on this unit. Two different units cannot have the same batch activation tag.			
ActiveType	This attribute defines when a batch begins (becomes Active) or ends (becomes Inactive) on a unit. The transition is a function of the data type and value of the batch activation tag. The two possible types are *Pulse* (default) and *Step*.			
	PULSE	The Inactive / Active transition occurs according to the following table. A change in value within the same status range has no effect on the status of the unit.		
		ActiveTag data type	*Batch becomes Inactive*	*Batch becomes Active*
		Integer	Value = 0	Value <> 0
		Digital	Value = first digital state	Value <> first digital state

Attribute	**Description**			
		Float	-1 < Value < 1	Value <= -1 or Value >= 1
		String	Value = ""	Value <> ""
	STEP	Units with this type are normally active: every time the ActiveTag receives a new value, the current batch is completed and a new batch is started. New batches will cease being created for each new value when the ActiveTag receives the *Stop Value*.		
		ActiveTag data type	*Stop Value*	
		Integer	Value = 0	
		Digital	Value = first digital state	
		Float	-1 < Value < 1	
		String	Value = ""	
BIDExpr	This defines the expression consisting of PI Tags and text to generate a batch identifier (BatchID) when a batch starts on a unit. See the section, *Rules for Defining BIDExpr and ProdExpr,* for more information.			
ProdExpr	This defines the expression consisting of PI tags and text to generate a product identifier (ProductID) when the batch starts on a unit. See the section, *Rules for Defining BIDExpr and ProdExpr,* for more information.			
EvalDelay	Delay (default is zero), in seconds, to wait before evaluating BIDExpr and ProdExpr after the batch starts. If the batch duration is shorter than the evaluation delay, the BIDExpr and ProdExpr will be evaluated when the batch ends. This attribute is useful if the BatchID and ProductID are not defined until after the batch starts.			
MergeConsecutive	If the value of this attribute is non-zero (default is zero) and the *ActiveType* is *Pulse*, consecutive batches with the same BatchID are considered to be one batch. The first assigned *ProductID* is used for the entire batch. This attribute is useful for batches that are halted temporarily and then restarted before the actual batch completes.			
Description	A textual description of the unit.			
DataAccess	Similar to tag security, this security attribute controls access to the batch data for the unit for the owner, group, and world. The format is the following: "o:<access> g:<access> w:<access>" where the <access> mask is either "rw" for read-write access, "w" for write-only access, "r" for read-only access, or "" for no access. The default access string is "o:rw g:r w:r".			
DataGroup	This security attribute indicates which group of PI users can access batch data for the unit. The default data group is *piadmin*.			
DataOwner	This security attribute indicates which individual PI user owns the batch data for the unit. The default data owner is *piadmin*.			
UnitAccess	Same as *DataAccess*, except this controls access to the configuration for the unit. The default access string is also "o:rw g:r w:r".			
UnitGroup	This security attribute indicates which group of PI users can access the configuration for the unit. The default unit group is *piadmin*.			

Attribute	Description
UnitOwner	This security attribute indicates which individual PI user owns the configuration for the unit. The default unit owner is *piadmin*.

Rules for Defining BIDExpr and ProdExpr

If only one type of product is made on a given unit, a simple fixed text string will suffice for **ProdExpr**. Typically, however, a given unit supports many different products, so the ProductID must be generated from the value of one or more PI tags when the batch begins. Similarly, a different BatchID will likely need to be generated even though it is not required to be unique.

To generate multiple BatchIDs or ProductIDs requires an expression similar to the following:

```
″ ’R1:PRODA’/4 + ¥″_¥″ + ’R1:PRODB’/5 + ¥″-A¥″ ″
```

Note: The backslashes preceding the inner double quotes are needed; otherwise, they would be misinterpreted to be the final double quote of the entire expression.

Such an expression combines the values of the PI tags *R1:PRODA* and *R1:PRODB* with some additional text to generate a product name. Some example values of the PI tags and the resulting ProductID (or BatchID) are:

R1:PRODA	R1:PRODB	ProductID
5	765.99	0005_00765-A
BLACK	YELLOW	BLAC_YELLO-A
ABC	XYZ_X	ABC _XYZ_X-A
12345	Shutdown	####_Digital State Set-A

The following is a complete list of the syntax rules and limitations for specifying both **BIDExpr** and **ProdExpr** expressions:

- ❑ The entire expression must be enclosed in double quotes.
- ❑ PI points must be enclosed in single quotes.
- ❑ Static text strings must be enclosed in double quotes. Backslashes must precede the double quotes surrounding text strings to prevent misinterpretation of the final double quote for the entire expression.
- ❑ A number following the field size indicator, '/', indicates the number of characters the field will occupy. A value of **0** or an unspecified field width indicates that all characters are required.
- ❑ The values for integer tags are right-justified and zero-filled if the number of digits is less than the field size. If the number of integer digits exceeds the field size, the field is filled with '#'s.

- ❑ The values for float tags are truncated and then converted to integers, which means the field size rules for integers then apply.
- ❑ The values for digital or string tags are left-justified and filled with trailing spaces if the number of characters is less than the field size. If the number of characters exceeds the field size, the text is truncated.
- ❑ If any tag has a bad status, the field is filled with '#'s.
- ❑ Any leading or trailing blanks in the resulting BatchID or ProductID are truncated.
- ❑ The resulting BatchID or ProductID should not contain any of the following characters: *** ?** or \. These characters will interfere with wildcard searches.

Support for Web Processes

The PI Batch Subsystem supports batches for paper machines and other web processes like textiles and film. For these types of batches, the batch active tag should be of type **Step**. This tag may be used to record the sequential batch number on a particular machine (unit), taking on a new value when a new batch is started. The batch start time will be the timestamp of the value of the tag, and the batch end time will be one second before the timestamp of the next value in the Archive.

Configuration Differences from PI BA in PI2 (OpenVMS)

The PI Batch Subsystem configuration differs from PI BA in PI2 (OpenVMS). The differences and the reasons for the differences are summarized in Table 6–2.

Table 6–2. Configuration Differences from PI BA in PI2 (OpenVMS)

Difference from PI BA in PI2	Reason
Removal of the list of TAG aliases for the UNIT.	Tag aliases are included in the PI3 Server. Therefore, they are not duplicated in the PI Batch Subsystem.
Removal of the STEP and CYCLETIME tags.	These functions are not supported in the PI Batch Subsystem. Instead, this functionality is provided by Performance Equations.
Addition of the type of the batch active tag	The starting and stopping of a batch on a unit is indicated by the value of the unit’s batch active tag. ▪ For PI BA in PI2, all of the batch active tags were of type *pulse*. That is, when the value of the batch active tag is 0 (or its digital equivalent) the unit is considered inactive. When the batch active tag is 1 (or its digital equivalent), the unit is considered active. ▪ Batch active tags of type *Step* are not restricted to the values of 0 and 1. To support web processes, described above, the batch active tag takes on a new value when a batch is started, ending the previous batch.

Common Operations

The following sub-sections demonstrate the common tasks of *creating*, *listing*, *renaming*, *editing*, and *deleting* units.

Create Units

Recall that the only attributes with default values are the following: **ActiveType**, **EvalDelay**, **MergeConsecutive**, and the six security attributes (**DataAccess**, **DataGroup**, **DataOwner**, **UnitAccess**, **UnitGroup**, **UnitOwner**). If the default values for these attributes are acceptable, then these attributes do not have to be specified when creating new units. The following **piconfig** commands will create the unit *TestUnit* using the default attribute values:

```
* (Ls - ) PIconfig> @table PIBAUNIT
* (Ls - PIBAUNIT) PIconfig> @mode create
* (Cr - PIBAUNIT) PIconfig> @istr unitname,activetag,bidexpr,prodexpr,
description
* (Cr - PIBAUNIT) PIconfig> TestUnit,TestUnitTrigger,"¥"TestUnitBID¥"",
"¥"TestUnitPID¥"","This is a test unit."
*> TestUnit,TestUnitTrigger,"¥"TestUnitBID¥"","¥"TestUnitPID¥"","This is a test
unit."
```

When a new unit is created, a unique PI point (e.g. *piba36*) is created to store batch data in the Archive just for that particular unit. One easy way to determine the name of this point, which may be needed during troubleshooting, is to examine the PI Server Message Log at the time the unit is created for a message similar to:

```
0 pipoints 27-Nov-05 22:46:27
>> Point Added by User (1) piadmin, piba36, PtId: 8406, RecNo: 4283
```

List Units

Use the following **piconfig** commands to list all the attributes for *TestUnit*:

```
* (Ls - ) PIconfig> @table pibaunit
* (Ls - PIBAUNIT) PIconfig> @mode list
* (Ls - PIBAUNIT) PIconfig> @ostr *
* (Ls - PIBAUNIT) PIconfig> @select unitname=TestUnit
* (Ls - PIBAUNIT) PIconfig> @ends
TestUnit,TestUnitTrigger,pulse,"TestBID",o:rw g:r w:r,piadmin,piadmin, This is
a test unit.,0,0,"TestPID",o:rw g:r w:r,piadmin,piadmin
```

Rename Units

Renaming units requires the use of the *NEWUnitName* attribute in edit mode. The following **piconfig** commands rename *TestUnit*:

```
* (Ls - ) PIconfig> @table pibaunit
* (Ls - PIBAUNIT) PIconfig> @mode edit
* (Ed - PIBAUNIT) PIconfig> @istr unitname,newunitname
* (Ed - PIBAUNIT) PIconfig> TestUnit,TestUnit1
*> TestUnit,TestUnit1
```

Because a unit's name is not used to store the actual batch data, renaming a unit will be wholly transparent to batch data searches and retrieval.

Edit Units

Any unit attribute except for **UnitName** can be edited directly using the attribute's name. The following **piconfig** commands will edit the security attributes **DataAccess** and **UnitAccess** for *TestUnit*:

```
* (Ls - ) PIconfig> @table pibaunit
* (Ls - PIBAUNIT) PIconfig> @mode edit
* (Ed - PIBAUNIT) PIconfig> @istr unitname,dataaccess,unitaccess
* (Ed - PIBAUNIT) PIconfig> TestUnit,"o:rw g:rw w:","o:rw g:rw w:"
*> TestUnit,"o:rw g:rw w:","o:rw g:rw w:"
```

Editing a unit will also result in editing of the unique PI point (e.g. *piba36*) used to store the unit's batch data in the archive. Examining the PI Server message log around the time of a unit edit should yield a message similar to:

```
0 pipoints 27-Nov-05 23:17:22
>> Point Edited by User (1) piadmin, piba36, Point Id: 8406
```

Delete Units

Rockwell Automation does not recommend deleting units, especially those that have valid batch data. Nevertheless, the following **piconfig** commands will delete *TestUnit*:

```
* (Ls - ) PIconfig> @table pibaunit
* (Ls - PIBAUNIT) PIconfig> @mode delete
* (Dl - PIBAUNIT) PIconfig> @istr unitname
* (Dl - PIBAUNIT) PIconfig> TestUnit
*> TestUnit
```

In case of user error, deleting a unit will not automatically result in the deletion of the unique PI point used to store the unit's batch data in the Archive. Instead, the batch data storage point will simply be renamed. Examining the PI Server Message Log around the time of a unit deletion should yield a message similar to:

```
0 pipoints 27-Nov-05 23:30:04
>> Point Renamed by User (1) piadmin, New Tag piba36_del, Old Tag piba36,
PointId: 8406
```

To completely delete the unit and all of its batch data would require the second manual step of deleting the renamed batch data storage point (e.g. *piba36_del*) for that particular unit.

6.3.2 Alias Configuration

PI points are often named to reflect the instrument data source in order to provide an obvious mapping between the two. However, in practice, except possibly for the instrument or process engineers, it is much easier to reference a particular attribute by its unit name and common name. Furthermore, in many cases, the unit name is implied, so the common name itself is an unambiguous reference to the physical attribute.

For example, as shown below, a plant may have three very similar reactors with the same three important attributes: level, temperature, and flow. The corresponding PI points for the attributes would be different for each reactor.

Unit Name	Common Attribute Name	PI Point Name
Reactor1	Level	LIC:129732.PV
Reactor1	Temperature	TIC:135432.PV
Reactor1	Flow	FIC:245433.PV
Reactor2	Level	LIC:297324.PV
Reactor2	Temperature	TIC:254326.PV
Reactor2	Flow	FIC:245432.PV
Reactor3	Level	LIC:397321.PV
Reactor3	Temperature	TIC:354399.PV
Reactor3	Flow	FIC:345439.PV

Aliases in the PI Batch Subsystem provide the mechanism to enable the more natural reference to an attribute's common name instead of the more obscure instrument name.

PI Batch Alias (PIBAALIAS) Table

The configuration for all aliases is stored in the data file *\PI\dat\pibaalias.nt* and is accessed exclusively via the **piconfig** table named **PIBAALIAS.** Compared to unit configuration, alias configuration is very simple, involving only a couple of attributes listed below. The list includes each attribute's name, data type, default value ("**D**:"), and current value ("**C**:").

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @?atr
1 - Alias String D: C:
2 - NEWAlias String D: C:
3 - Tag String D: C:
```

A complete description of each of the alias attributes appears below in Table 6–3.

Table 6–3. PIBAALIAS Table Attributes

Attribute	Description
Alias	The alias name has two components: unit name and common name. The syntax for an alias is the following: `¥¥unit name¥common name`. The unit name must correspond to an existing unit in the PIBAUNIT table.
NEWAlias	Used to rename an existing alias.
Tag	The name of the PI point associated with the alias. The PI point must already exist

Common Operations

The following sub-sections demonstrate the common tasks of *creating*, *listing*, *renaming*, *editing*, and *deleting* aliases.

Create Aliases

The following piconfig commands will create three aliases on unit *Reactor1*, one for each of the attributes level, temperature, and flow:

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @mode create
* (Cr - PIBAALIAS) PIconfig> @istr alias,tag
* (Cr - PIBAALIAS) PIconfig> ¥¥Reactor1¥level,LIC:129732.PV
*> ¥¥Reactor1¥level,LIC:129732.PV
* (Cr - PIBAALIAS) PIconfig> ¥¥Reactor1¥temperature,TIC:135432.PV
*> ¥¥Reactor1¥temperature,TIC:135432.PV
* (Cr - PIBAALIAS) PIconfig> ¥¥Reactor1¥flow,FIC:245433.PV
*> ¥¥Reactor1¥flow,FIC:245433.PV
```

List Aliases

Because aliases include the unit name, a wildcard (default: '*') is often used to select or list specific aliases. The following **piconfig** commands list all aliases defined for *Reactor1*:

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @mode list
* (Ls - PIBAALIAS) PIconfig> @ostr alias,tag
* (Ls - PIBAALIAS) PIconfig> @select alias=¥¥Reactor1¥*
* (Ls - PIBAALIAS) PIconfig> @ends
¥¥Reactor1¥flow,FIC:245433.PV
¥¥Reactor1¥level,LIC:129732.PV
¥¥Reactor1¥temperature,TIC:135432.PV
```

The wildcard can be used to list a variety of other alias combinations. For example, `@select alias=*` will list all aliases for all units; `@select alias=¥¥*¥temperature` will list all aliases with a common name of *temperature*.

Rename Aliases

Renaming of aliases is performed in edit mode using the **NEWAlias** attribute. The following **piconfig** commands rename the temperature alias of *Reactor1*:

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @mode edit
* (Ed - PIBAALIAS) PIconfig> @istr alias,newalias
* (Ed - PIBAALIAS) PIconfig> ¥¥Reactor1¥temperature,¥¥Reactor1¥CoreTe
mp
*> ¥¥Reactor1¥temperature,¥¥Reactor1¥CoreTemp
```

Edit Aliases

Edit mode is also used when changing the corresponding PI point for an alias. The following **piconfig** commands change the PI tag associated with the temperature alias of *Reactor1*:

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @mode edit
* (Ed - PIBAALIAS) PIconfig> @istr alias,tag
* (Ed - PIBAALIAS) PIconfig> ¥¥Reactor1¥temperature,TIC:234531.PV
*> ¥¥Reactor1¥temperature,TIC:234531.PV
```

Delete Aliases

The following **piconfig** commands delete the temperature alias of *Reactor1*:

```
* (Ls - ) PIconfig> @table PIBAALIAS
* (Ls - PIBAALIAS) PIconfig> @mode delete
* (Dl - PIBAALIAS) PIconfig> @istr alias
* (Dl - PIBAALIAS) PIconfig> ¥¥Reactor1¥temperature
*> ¥¥Reactor1¥temperature
```

6.4 Batch Data Information

All batch data for all units is stored in the Data Archive. When a unit is created, it is assigned a unique Archive point named *pibaN*, where ***N*** is the unit's unique integer ID, for storing batch data specific to that unit. Each completed batch consists of two events written to the appropriate *pibaN* point: an event at the start time of the batch with a System Digital State of *ActiveBatch*; and an event at the end time of the batch with the actual batch data stored as a Blob object.

6.4.1 PI Batch Data (PIBATCH) Table

The batch data events should typically never be accessed directly from the archive. Instead, batch data should be accessed via the **piconfig** table named **PIBATCH**. The below list includes the name, data type, default value (**D**:), and current value (**C:**) of each of the batch data attributes.

```
* (Ls - ) PIconfig> @table PIBATCH
* (Ls - PIBATCH) PIconfig> @?atr
  1 - UnitName        String  D:                    C:
  2 - NEWUnitName     String  D:                    C:
  3 - BID             String  D:                    C:
  4 - BIDsearch       String  D:                    C:
  5 - Count           String  D: 50                    C:
  6 - Handle          String  D:                    C:
  7 - ProdID          String  D:                    C:
  8 - ProdIDsearch    String  D:                    C:
  9 - SearchStart     String  D:                    C:
 10 - SearchStop      String  D:                    C:
 11 - StartStatus     String  D:                    C:
 12 - StartTime       String  D:                    C:
 13 - StopStatus      String  D:                    C:
 14 - StopTime        String  D:                    C:
```

A complete description of each of the batch data attributes appears below in Table 6–4.

Table 6–4. PIBATCH Table Attributes

Attribute	Definition
UnitName	The name of the unit associated with a batch. In searches, the wildcard search string for unit names.
NEWUnitName	This attribute is not supported.
BID	The name or identifier of a batch. This is evaluated when a batch starts (+ evaluation delay).
BIDsearch	The wild card search string for BatchIDs.
Count	The maximum number of batches to retrieve in a search. If not specified, the default number of batches returned is 50.
Handle	The unique identifier for a single batch. The format is the following: <uniqueID>-<StopTime in UTC>. The unique ID is the same integer (***N***) used in the name *piba**N*** of the unit's Archive point.
ProdID	The name or identifier of a batch's product. This is evaluated when a batch starts (+ evaluation delay).
ProdIDsearch	The wild card search string for ProductIDs.
SearchStart	The start time of a search.
SearchStop	The end time of a search.
StartStatus	An integer indicating the status of a batch's start time: 0 = Not Set; 1 = Unknown; 2 = OK.
StartTime	The time a batch started.
StopStatus	An integer indicating the status of a batch's end time: 0 = Not Set; 1 = Unknown; 2 = OK.
StopTime	The time a batch completed.

6.4.2 Common Operations

The following sub-sections demonstrate the common tasks of *creating*, *listing*, *editing*, and *deleting* batches.

Create Batches

New batches may only be created for existing units, and the start and stop times are subject to the following three conditions:

- ❑ The start time must be before the stop time.
- ❑ The time span of the batch must not overlap an existing batch on the same unit.
- ❑ There must be a registered Archive that covers the time span of the batch.

BID and **ProdID** are optional but typically very useful attributes. A Handle should not be specified, for it will be generated. The following **piconfig** commands will create a batch for *TestUnit2*:

```
* (Ls - ) PIconfig> @table pibatch
* (Ls - PIBATCH) PIconfig> @mode create
* (Cr - PIBATCH) PIconfig> @istr unitname,bid,prodid,starttime,stoptime
* (Cr - PIBATCH) PIconfig> TestUnit2,"Batch-1","Prod-1","24-Dec-05 04:00","24-
Dec-05 08:00"
*> TestUnit2,"Batch-1","Prod-1","24-Dec-05 04:00","24-Dec-05 08:00"
```

List Batches

Recall that the following five attributes can be specified as search input to narrow the list of batches: **UnitName**, **BIDSearch**, **ProdIDSearch**, **SearchStart**, and **SearchStop**. A typical search will specify at least **UnitName**, **SearchStart**, and **SearchStop**. The following **piconfig** commands will list all batches for *TestUnit2* since December 1, 2005:

```
* (Ls - ) PIconfig> @table pibatch
* (Ls - PIBATCH) PIconfig> @mode list
* (Ls - PIBATCH) PIconfig> @ostr unitname,bid,prodid,starttime,
stoptime,handle
* (Ls - PIBATCH) PIconfig> @ostr ...
* (Ls - PIBATCH) PIconfig> @istr unitname,searchstart,searchstop
* (Ls - PIBATCH) PIconfig> TestUnit2,1-dec-05,*
*> TestUnit2,1-dec-05,*
TestUnit2,Batch-1,Prod-1,24-Dec-05 04:00:00,24-Dec-05 08:00:00,35-1135440000
TestUnit2,Batch-2,Prod-2,24-Dec-05 08:01:00,24-Dec-05 12:00:00,35-1135454400
* End Repeat...
```

Note: The command `@ostr …` is required to force the output of multiple batches. Without this command, only the first batch that matches the input criteria will be output.

When listing batches, the **Count** attribute must also be taken into consideration. If **Count** is not specified in the input structure, then a maximum of 50 batches will be output.

For user convenience, the installed **piconfig** script *\PI\adm\pibalist.dif* specifies the same attributes in the input and output structures as those used in the above example, the typical search.

Edit Batches

Any combination of the following four batch attributes may be edited: **StartTime**, **StopTime**, **BID**, **ProdID**. Modification of **StartTime** or **StopTime** is subject to the same three conditions identified in *Create Batches*, and if the time modification is valid, the respective time status attribute will always be set to **2 = "OK"**.

In addition to the attributes that will be modified, the batch Handle must also be specified to uniquely identify the target batch for modification. Because a Handle encapsulates **StopTime**, modification of **StopTime** will render that batch Handle obsolete after the edit.

The following **piconfig** commands modify the **BID** and **StopTime** of the last December batch for *TestUnit2*:

```
* (Ls - ) PIconfig> @table pibatch
* (Ls - PIBATCH) PIconfig> @mode edit
* (Ed - PIBATCH) PIconfig> @istr handle,bid,stoptime
* (Ed - PIBATCH) PIconfig> 35-1135454400,"Batch-2Mod","24-Dec-05 14:00:00"
*> 35-1135454400,"Batch-2Mod","24-Dec-05 14:00:00"
```

Delete Batches

As when editing batches, the Handle must be specified when deleting batches. After a delete, the Handle will always be rendered obsolete. No other attributes need to be specified. The following **piconfig** commands delete the last December batch for *TestUnit2* (note the Handle changed after modifying **StopTime** in the *Edit Batches* example):

```
* (Ls - ) PIconfig> @table PIBATCH
* (Ls - PIBATCH) PIconfig> @mode delete
* (Dl - PIBATCH) PIconfig> @istr handle
* (Dl - PIBATCH) PIconfig> 35-1135461600
*> 35-1135461600
```

6.5 Batch Subsystem Operation

The following sections describe the primary operating tasks of the Batch Subsystem.

6.5.1 Check for Unit Consistency

When the Batch Subsystem first starts up, it checks all defined units (stored in *\PI\dat\pibaunit1.nt*) for consistency. The consistency check includes verifying the existence of the batch activation tags as well as any tags referenced in the expressions for generating BatchID and ProductID. Any unit indicated as active is also checked to verify that the same batch is still running, and if there is a discrepancy, the Data Archive is updated appropriately. Similarly, if any unit previously indicated as *inactive* is detected to have an actively running batch, the Data Archive is updated accordingly as well. If any problems are encountered, messages are sent to the PI Message Subsystem.

6.5.2 Monitor Activation Tags for Transitions

After startup is complete, the Batch Subsystem begins monitoring all batch activation tags for exception reports. The Batch Subsystem will remain in this state until it is shut down. If the Batch Subsystem detects a transition for a batch activation tag from an *inactive* to *active* state, the unit is updated to record the start of the batch. Until batch completion, all the information for a running batch is actually stored with the unit (in *\PI\dat\pibaunit1.nt*). If an *active* to *inactive* state transition is detected, the batch is considered complete, and the Blob object containing the complete batch structure is created and finally stored in the Data Archive.

6.5.3 Evaluate BIDExpr and ProdExpr

The Batch Subsystem normally evaluates the **BIDExpr** and **ProdExpr** to generate the BatchID and ProductID, respectively, when the batch activation tag transitions from an *inactive* to *active* state. The evaluation of these expressions can be delayed for a unit by using the **EvalDelay** attribute. After the evaluation time, changes to any tags involved in a **BIDExpr** and **ProdExpr** are ignored.

6.6 Client Access to Batch Subsystem Batches

Other than with **piconfig**, read-only programmatic access to the batches created by the Batch Subsystem is available directly with the PI-API. Read-only programmatic access is also available via the PI-SDK, but only if the units are first registered with the Batch Database; this registration procedure is detailed in the document ***PI Batch Database Support of the PI Batch Subsystem***.

The only API-based client application is the PI BatchView add-in for Excel, but of course, custom API applications can always be written. All other client applications such as BatchView for FactoryTalk Historian ProcessBook or the stand-alone BatchView Quick Search are SDK-based applications, and therefore require unit registration with the Batch Database. See corresponding user guides for more information.

6.7 Complete Example

Installation of the Batch Subsystem does not result in the creation of any units or batches. The installed **piconfig** script *\PI\adm\pibatch.dif* creates a sample unit and enable the automatic creation of sample batches. Of course, the actual configuration and batch data is contrived, but the example provides a complete functional overview of the Batch Subsystem.

Before executing the script, ensure that the Batch Subsystem and all other PI Server subsystems are fully started. To execute the script, simply enter the following at a command prompt after changing to the *\PI\adm* directory:

```
piconfig < pibatch.dif
```

The script first creates two digital sets to model BatchIDs and ProductIDs and two digital tags, one assigned to each set, to be used in the **BIDExpr** and **ProdExpr** attributes of the sample unit. These digital tags are configured to be serviced by the Random Simulator Interface (executable name: **random**), a default interface distributed with all PI Servers, so this interface also needs to be configured and started for the example to work properly.

The script then creates a single sample unit named *Reactor1* using the default point *BA:Active.1* as the batch activation tag. If the default points were not created at PI Server installation time, then they can be created manually by executing the following at a command prompt after changing to the *\PI\adm* directory:

```
   piconfig input defaultpt.dif input defaultpt.dat exit
```

BA:Active.1 is serviced by the Ramp Soak Simulator Interface (executable name: **rmp_sk**), another default interface distributed with all PI Servers, so this interface also needs to be configured and started for the example to work properly.

The script finishes with the creation of a few aliases–*Temperature*, *Level*, and *Concentration*–using other default points–*BA:Temp.1*, *BA:Level.1*, and *BA:Conc.1*, respectively.

If the script executed without errors and all the simulator interfaces are running, a new batch for *Reactor1* should be created approximately every 81 minutes.

Chapter 7. PI Totalizer Subsystem

The **PI Totalizer Subsystem** (**Totalizer**) performs common calculations such as totals, averages, minimum and maximum values, and standard deviations. Output of a calculation is stored in a PI point.

The main difference between a Performance Equations point and a Totalizer point calculating the same summary is that Totalizer calculates from realtime inputs (as opposed to archived values.) Performance Equations are based on Archive events, while Totalizer results are based on Snapshot events.

PI Totals are the most accurate way to represent production summary data. Totalizers can be started and reset based on time and event, and ensure the highest accuracy in the calculation of flow volumes and other critical variables used to monitor product transfers or production performance. Totalizer is especially practical for totaling measurements or other process variables at the end of specific time periods, such as the end-of-day yields.

This chapter includes the following topics:

7.1 Totalizer Subsystem Overview

Totalizer allows you to perform certain calculations on a point in the Snapshot, and to store the results in another point. The process is called *postprocessing*. Postprocessing includes the following types of summary calculation:

- Total
- Average
- Minimum
- Maximum
- Range

- Standard Deviation
- Median

Additionally, Totalizer permits the counting of update events for a point. The types of counting allowed are as follows:

- All Events
- Event Equal To a value
- Event Not Equal To a value
- Event Greater Than a value
- Event Greater Than or Equal To a value
- Event Less Than a value
- Event Less Than or Equal To a value
- Event change from Greater Than or Equal To to Less Than
- Event change from Less Than to Greater Than or Equal To

The Totalizer is a dedicated subsystem, **pitotal**. This subsystem signs up for exceptions, which means that it is notified when a new value is added to the Snapshot for any of the points to be postprocessed. After postprocessing, values, for example, **average**, **total**, or **time in state**, are sent back to the PI Snapshot.

Note: Since **pitotal** uses Snapshots of the data, postprocessing uses uncompressed data. Its results are more accurate than those computed later from values in the archive.

7.1.1 Totalizer vs. Performance Equations

Totalizer may be more accurate because the values used in Totalizer calculations are taken from the Snapshot, not after the application of compression as in the case of Performance Equations. Figure 7–1 shows an interval of time for which an accumulation of a point occurs. Within this accumulation interval, the point also undergoes exception reporting.

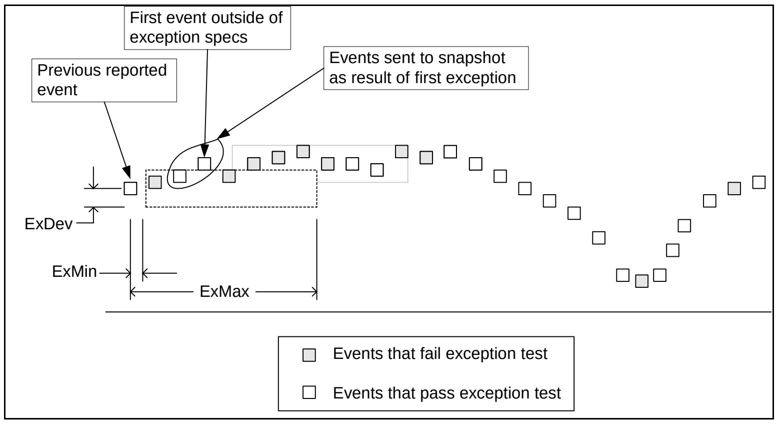

First event outside of exception specs
Events sent to snapshot as result of first exception
Previous reported event
ExDev
ExMin
ExMax
Events that fail exception test
Events that pass exception test

Figure 7–1. Exception Reporting

The values are sent to the Snapshot if they pass the exception test defined by the **ExDev**, **ExMin** and **ExMax** attributes. In Figure 7–2, 10 of the 28 values fail the exception test and are not sent to the Snapshot. The Totalizer utilizes the 18 values that are left for the evaluation. On the other hand, the Performance Equation Scheduler uses the compressed values (if compression is on). Figure 7–2 shows the difference in the resultant time-weighted calculation for the Totalizer and the Performance Equation Scheduler.

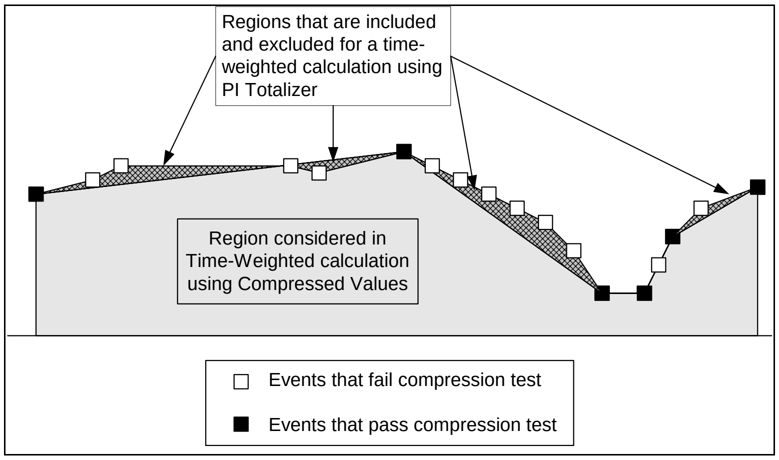

Regions that are included and excluded for a time-weighted calculation using PI Totalizer
Region considered in Time-Weighted calculation using Compressed Values
□ Events that fail compression test
■ Events that pass compression test

Figure 7–2. Time Weighted Calculation for Totalizer and Performance Equation

The region considered for the calculation is the area under the curve created by connecting the 18 values left from the exception test. On the other hand, the region considered by the Performance Equation calculation would be the area under the curve that is created by connecting the six points that are left after the compression test. In general, the difference in accuracy between the two calculations is about 1 to 2.5 percent.

The Totalizer may also be more efficient in many cases, especially when the summary interval is large. Consider a year-to-day calculation. If the calculation is implemented as a performance equation, the Archive data from the beginning of the year to the current time needs to be retrieved to carry out the calculation. On the other hand, if the calculation is implemented as a Totalizer point, then only the new Snapshot event is needed to do the calculation.

7.2 Totalizer ConfigurationOverview

Totalizer is very flexible, offering a wide range of operating modes. The behavior of a particular point is primarily determined by five multiple-choice parameters:

Table 7–1. Five Major Parameters that Affect Totalizer

RateSampleMode	**CalcMode**	**Function**
Natural	*TimeWeighted*	*Total*
Scan1	*EventWeighted*	*Average*
Scan2	*AllEvents*	*Minimum*
Event	*EventChange*	*Maximum*
	TimeTrue	*Range*
TotalCloseMode	**ReportMode**	*StdDeviation*
		Median
Clock	*PeriodEnd*	*EventEQ*
EventChange	*Ramping*	*EventNE*
EventTrue	*Running*	*EventGE*
NSampleMoving	*RunEstimate*	*EventGT*
NSampleBlock	*RunEst2*	*EventLE*
TimeMoving		*EventLT*
Forever		*EventGE_LT*
		EventLT_GE

Not counting the behavior influences of the **Rate Point Data Type** and **Step** parameter, the number of combinations of the five parameters is 12,000 (5 x 5 x 6 x 16 x 5). However, not all of the combinations are allowed. For example, a point that is computing a total with a **TotalCloseMode** option of *Forever* does not allow the **ReportMode** of *PeriodEnd* because the totalization cannot stop at the end of the period. The combinations of the allowed cases are described later in this chapter.

7.2.1 Creation of a Totalizer Point

Totalization involves several steps, as follows:

1. Sample the value (**RateSampleMode**) from the input point called the *rate point* (SourceTag).
2. Possibly exclude the value based on a filter (**Filter Expression**).
3. Accumulate the observed values as needed for the specified summary function (**Function** and **CalcMode**).
4. Determine when the Totalizing period is complete and the accumulators need to be reset (**TotalCloseMode**).
5. Report the result (**ReportMode**).

Figure 7–3 shows a flow diagram for the steps that are involved in creating a Totalizer point.

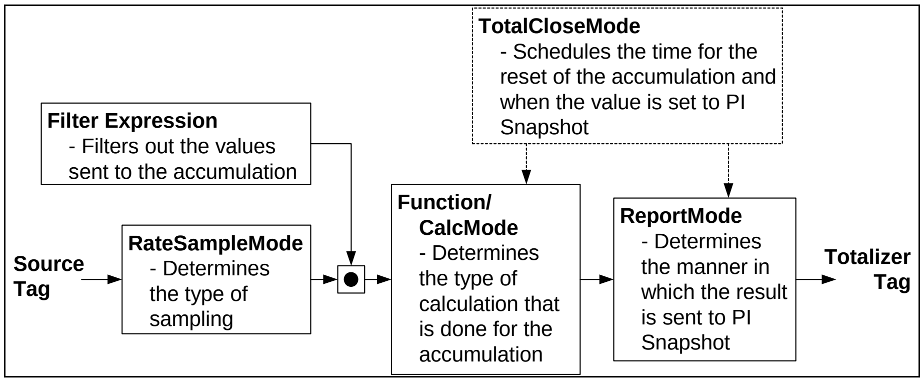

TotalCloseMode
- Schedules the time for the reset of the accumulation and when the value is set to PI Snapshot
Filter Expression
- Filters out the values sent to the accumulation
Source Tag
RateSampleMode
- Determines the type of sampling
Function/ CalcMode
- Determines the type of calculation that is done for the accumulation
ReportMode
- Determines the manner in which the result is sent to PI Snapshot
Totalizer Tag

Figure 7–3. Flow Diagram for the Creation of a Totalizer Point

7.2.2 Totalizer Input Values

Each Totalizer point has a single input called the *source* or *rate point*. This is the point whose values are summed, counted, or otherwise accumulated to produce the required summary value. The arrival of a new rate point value triggers the accumulation and reporting functions of each Totalizer point. Most often this process will be “naturally” scheduled, that is every new value reported to the system will be processed into the ongoing accumulation.

Filtering

Each point may have one filter expression. This expression is evaluated each time that a new update value is received for each tag referenced in the expression. The results are stored in time order until the next rate point value is received.

Accumulation

The accumulation step takes the previous and current rate point values into account when adding to the accumulators. If a filter is specified, the stored filter expression results are also considered. The details of the accumulation depend on the specified functions. The available functions are: **total, average, minimum, maximum, range, standard deviation, count** (of digital states), and **timer**.

Accumulation Interval

The accumulation interval is the interval of time for which a postprocessing calculation occurs. Totals are often generated for intervals specified by a period and an initial start time. Alternatively, the interval may be defined by external events using a trigger expression in PE form.

Output

The output of the results is usually reported at the end of the accumulation interval. The output can also be estimated before the completion of the accumulation interval. A running result is also available.

7.3 Totalizer Point Class Attributes

The points to which the Totalizer writes its postprocessed values must be of the Totalizer Point Class. This class has both **Base** and **Totalizer** attributes. Table 7–2 lists the additional attributes in the Totalizer Point Class attribute set. These Totalizer-specific parameters drive the totalization process.

The data types for the five major attributes, **RateSampleMode**, **TotalCloseMode**, **ReportMode**, **Function**, and **CalcMode** as well as the **SourceTag**, **Options**, and **CompVal**, are strings and are not case-sensitive. The filter and event expressions use the Performance Equation syntax. **Period**, **Period2**, **Offset** and **Offset2** are relative timestamps while the rest of the attributes are real numbers.

Table 7–2. Totalizer Point Class Attribute Set

Point Attribute	Default Value or Valid Options	Notes
SourceTag		Point to be postprocessed. (See **Options**, below in this table, for SourceTag values.)
RateSampleMode	*Natural (default)*	Processes when new rate point value arrives.
	Scan1	Time based samples based on **Period2** and **Offset2**, interpolated between update events.
	Scan2	Time based samples based on **Period2** and **Offset2**, extrapolated from the value of the last seen event.
	Event	Triggered only when **EventExpr** changes.
TotalCloseMode	*Clock (default)*	Accumulation interval based on **Period** and **Offset** parameters.
	EventChange	Closes when **EventExpr** result changes value.
	EventTrue	Totalizer on while value of **EventExpr** is non-zero.
	NSampleMoving	Period based upon a fixed number of input values that is set with the **MovingCount** parameter.
	NSampleBlock	Period based upon a block of input values. The number of input values for the block is set with the **MovingCount** parameter.
	TimeMoving	Moving time window based on **Period**.
	Forever	Never resets unless no previous valid point can be found or **Options** attribute is settable (for total, event counting or time only).

Point Attribute	Default Value or Valid Options	Notes
ReportMode	*PeriodEnd*	No output until period end.
	Ramping	Running result update to PI Snapshot at the time of the update (sawtooth).
	Running (default)	Running result update to PI Snapshot at one second after the period start time.
	RunEstimate	Projected output based on current rate value.
	RunEst2	Projected output based on average value.
Function	*Total (default)*	Numeric rate points only.
	Average	Numeric rate points only.
	Minimum	Numeric rate points only.
	Maximum	Numeric rate points only.
	Range	Numeric rate points only.
	StdDeviation	Numeric rate points only.
	Median	Numeric rate points only.
	Events	Numeric, digital, or string points.
	EventEQ	Numeric or Digital points.
	EventNE	Numeric rate points only.
	EventGE	Numeric rate points only.
	EventGT	Numeric rate points only.
	EventLE	Numeric rate points only.
	EventLT	Numeric rate points only.
	EventGE_LT	Change counter from **>=** to **<** (numeric or Digital points).
	EventLT_GE	Change counter from **<** to **>=** (numeric or Digital points).
CalcMode	*TimeWeighted (default)*	Values used in calculation weighted in proportion to duration during period.
	EventWeighted	The value of each update carries the same weight.
	AllEvents	Count all events matching condition.
	ChangeEvents	Count only events that were changes of value from the previous update.
	TimeTrue	Time in seconds event condition is true.
ZeroBias	*0.0*	Rate point value is zero below this value.
Period	*+1h*	Period of postprocessing used for **TotalCloseMode**.

Point Attribute	**Default Value or Valid Options**	**Notes**
Offset	*+0m*	Offset from beginning of day of the start of Totalizer.
MovingCount	*2*	Number of input values for **NSampleMoving** or **NsampleBlock**.
Period2	*+2m*	Used for **RateSampleMode** in **Scan1** and **Scan2**.
Offset2	*+0m*	Used for **RateSampleMode** in **Scan1** and **Scan2**.
PctGood	*85*	Lower limit on the percentage of input data that needs to be good.
Conversion	*1*	Conversion factor multiplies the accumulation result.
FilterExpr		Performance equation expression that filters the rate point.
EventExpr		Performance Equation expression used in **TotalCloseMode** and **RateSampleMode**.
CompValue	*ON*	Comparison value for event counting and time conditions expressed in the **Function** parameter.
Options	Field contains zero or more keywords for optional **Functions**.	
	InFromTotalizer	The rate point value is from an external (DCS) Totalizer block.
	UnderIsBad	Under range condition on rate point is bad data (default is zero).
	NoClampZero	Enable output to be less than zero.
	CloseAtEvent	Event ends accumulation interval.
	Setable	**TotalCloseMode** of **Forever** can be set externally.
	OneAtStart	Report is one value at 1 second into of the period for the **ReportMode** of **PeriodEnd**.
	OneAtEnd	Report is one value at the end of the period for the **ReportMode** of **PeriodEnd**.
	UnderIsZero	Source tag values are **UnderRange** (status code) are considered to be at the source tag zero value.
	OverIsTop	Source tag values are **OverRange** (status code) are considered to be at the source tag zero+span value.
	SourceStat	Use a bad source tag status in place of "**Bad Total**".
	LimitBack	Limits the number of totalization periods that will be reported between source tag events.

7.3.1 SourceTag

The **SourceTag** attribute specifies the rate point, which is the primary input for the postprocessing. This rate point must already exist and must be specified when the Totalizer point is configured. The rate point may be any numeric, digital, or string point types. The value and timestamp used in the postprocessing calculation is taken when an update of the rate point arrives.

7.3.2 RateSampleMode

The **RateSampleMode** attribute specifies when the individual values from the rate point will be used in the calculation or will be added to the accumulation. The default option for the **RateSampleMode** is *Natural*.

Natural (default)

Natural is the default and most common option. Figure 7–4 shows the update values of the rate point. The values that are considered in the postprocessing calculation are the update values that lie within the accumulation interval.

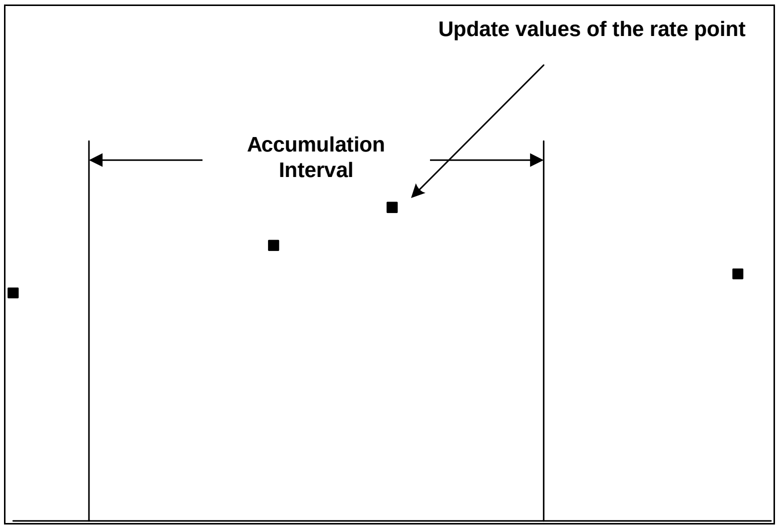


Figure 7–4. Natural Sampling

Depending on the other attributes such as the **TotalCloseMode**, **ReportMode**, **Function**, and **CalcMode**, the calculation of the result may or may not involve the update values of the rate point outside the accumulation interval. These features are discussed later in the chapter.

Scan1

Scan1 uses the attributes **Period2** and **Offset2** to sample the rate point at evenly spaced time intervals. Values that lie within two updates of the rate point values are linearly interpolated. Hence, **Scan1** does not compute the completed results until the rate point after the accumulation interval end time, as shown in Figure 7–5.

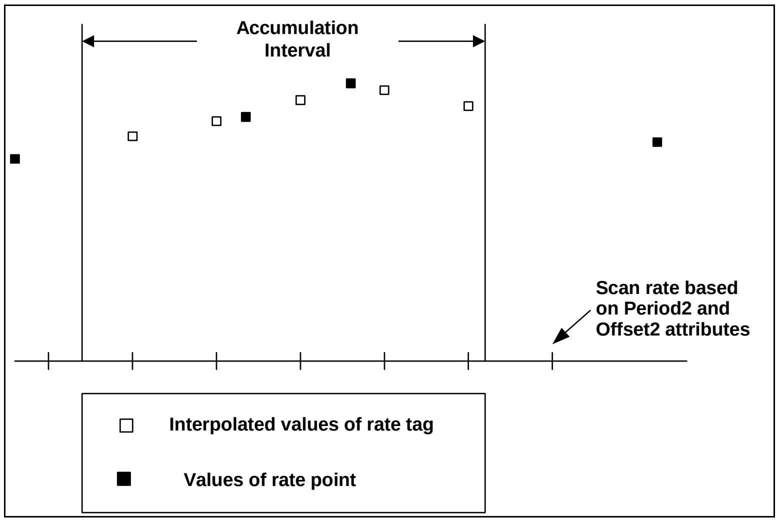

Accumulation Interval
Scan rate based on Period2 and Offset2 attributes
□ Interpolated values of rate tag
■ Values of rate point

Figure 7–5. Scan1 Sampling

The values used in the calculation are the interpolated values at the timestamps set using **Period2** and **Offset2**. Also, note that the sample rate may or may not coincide with the postprocessing period. For digital point counting, the digital state does not change until the update event of the rate point is a different digital state. Hence, *Scan1* will be similar to *Scan2* in that the digital state at a particular scan of the rate point will be the last seen digital state of the rate point.

Scan2

Scan2 is like *Scan1* in that it also samples at evenly spaced time intervals using the attributes **Period2** and **Offset2** except that sampled values are calculated from the last seen event. The value is projected almost as though the **Step** option was set. *Scan2* performs similarly to *Scan1* when counting digital states, as shown in Figure 7–6.

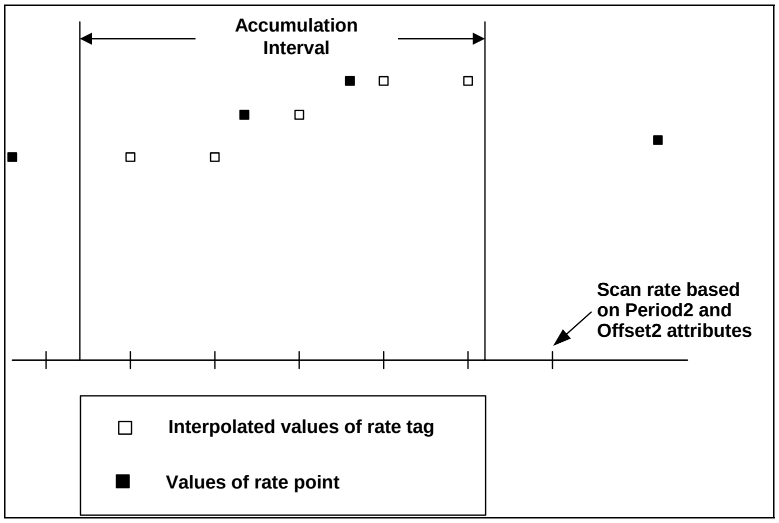

Accumulation Interval
Scan rate based on Period2 and Offset2 attributes
□ Interpolated values of rate tag
■ Values of rate point

Figure 7–6. Scan2 Sampling

Event

Event uses the event expression specified in the **EventExpr** attribute to determine when to sample the rate point. Sampling occurs when there is a change in the value of the event expression, as shown in Figure 7–7.

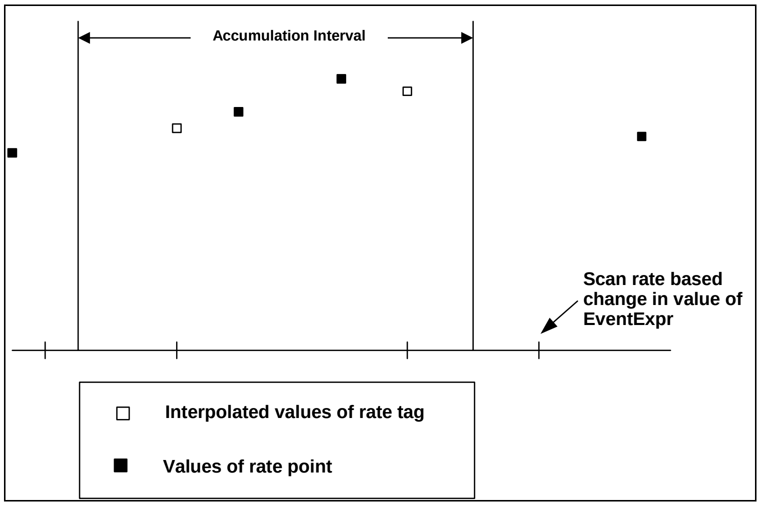

Accumulation Interval
Scan rate based change in value of EventExpr
□ Interpolated values of rate tag
■ Values of rate point

Figure 7–7. Event Sampling

The values used in the calculation are linearly interpolated between values of the rate point and timestamped at the time when the value of the event expression changes. *Event* performs similarly to *Scan1* in counting digital states.

7.3.3 TotalCloseMode

The option set in the **TotalCloseMode** attribute determines the accumulation interval. The Totalizer functions accumulate data over the accumulation interval. The calculation is “closed out” at the end of the accumulation interval and the “next event” occurs.

Note: “Closed out” means that the result is calculated and reported and the internal state of the accumulation function is reset. The “next event” refers to either a value of the rate point updating or an event from the event expression or the filter expression for the Totalizer point being updated.

Clock (default)

In **TotalCloseMode** of *Clock*, the Totalizer runs and resets at regular time intervals. The times are specified through two attributes, **Period** and **Offset**. **Period** is the length of each accumulation interval. **Offset** specifies the start of the first interval. The accumulation begins at the start of the day on which the Totalizer is started plus the amount of time given in the **Offset** parameter. **Period** is a relative time of any length. *Clock* is the default setting for this option.

The **Period** parameter may be specified as local time or UTC (Universal Coordinated Time). Periods longer than one hour that have no specifier are assumed to be local time. This allows totalization intervals to start at the same time every day even when the clocks are reset for Daylight Savings Time.

The **Offset** is a relative timestamp that is less than the **Period**. Figure 7–8 shows an example of the **Clock** option of the **TotalCloseMode**.

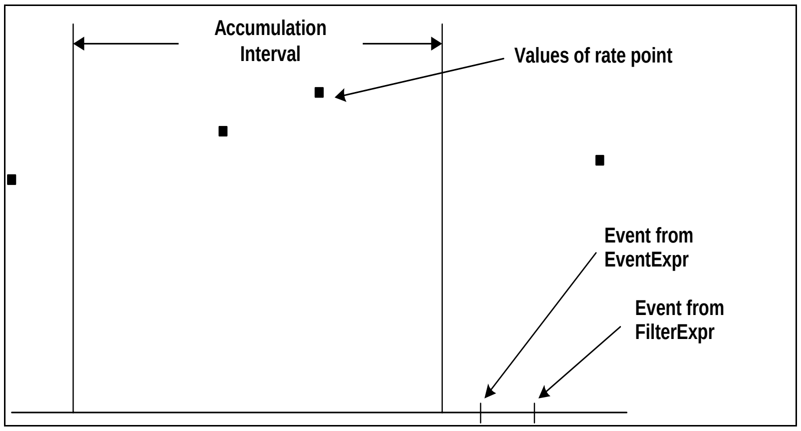


Figure 7–8. TotalCloseMode of Clock

In this example, although the accumulation ends, the actual close of the accumulation does not occur until the rate point updates. This event can occur well after the end of the accumulation interval. When the update event occurs, the calculated result is written to the Archive with timestamps of the actual totalization interval.

For the **RateSampleMode** of *Scan2*, the close out of the accumulation occurs at the first update event at or after the end of the period. In *Natural*, *Scan1* and *Event*, the closeout of the accumulation occurs only when the rate point value updates.

EventChange

This option connects the totalization period to external events. It requires a valid event expression be defined in the **EventExpr** attribute. The current accumulation will end and reset whenever the result of the event expression is different from that of the previous event expression calculation.

If the expression includes PI tags referencing the Snapshot, it is evaluated on a natural schedule. That is, it is recalculated whenever there is a new update for any of the tags. Expressions that do not reference Snapshot values are evaluated at times specified by the period and offset parameters but only when rate tag events beyond these times have been received. PI time functions may be used in these expressions to establish many useful intervals.

A naturally scheduled expression is only evaluated as each update is received for the real PI tags in the expression. The times of these events are used in the evaluation of any time functions in the expression. Tags that come from the same source (interface or remote PI system) as that of the rate tag should be used to assure synchronization of these critical timing events. The **Option**, *CloseAtEvent* may be used to force prompt closing of the Totalizer at the period end time. Time-weighted totals will assume the rate tag to be constant at the last value observed to the end of the period.

Expressions that contain no PI Snapshot references are evaluated at times specified by the **Period** and **Offset** parameters. These calculations are only performed when new updates on the rate tag are received. As each rate tag update arrives, its timestamp is compared with the next time the expression is due for calculation. If the due time is prior to the event timestamp, the expression is evaluated before the new event is fully processed.

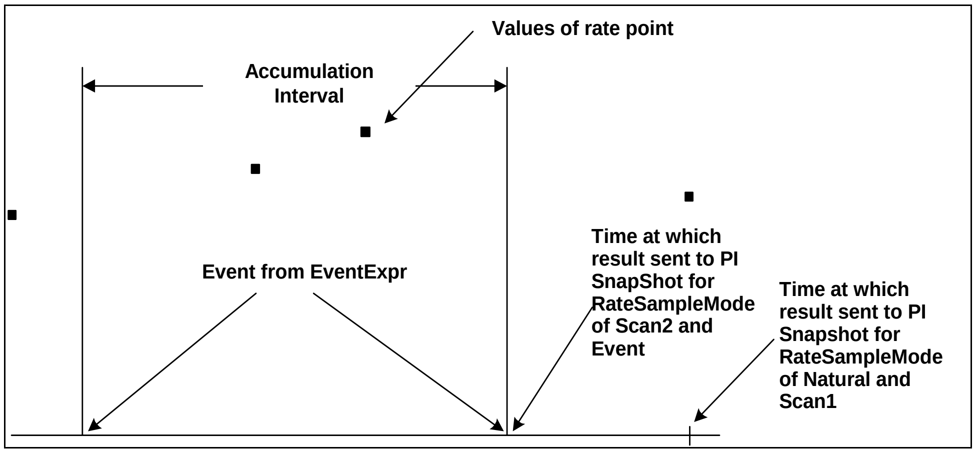

Values of rate point
Accumulation Interval
Event from EventExpr
Time at which result sent to PI SnapShot for RateSampleMode of Scan2 and Event
Time at which result sent to PI Snapshot for RateSampleMode of Natural and Scan1

Figure 7–9. TotalCloseMode of EventChange

As shown in Figure 7–9 for *Natural* and *Scan1*, the result of the accumulation will be delayed and not be sent to PI Snapshot until an update for the rate point is observed. If the event expression does not contain a point but only a time function, it will also be evaluated with the **RateSampleMode** of *Natural* based on the rate point.

EventTrue

This option also requires a valid event expression to be defined in the **EventExpr** attribute. In this case, as shown in Figure 7–10, an accumulation interval is started when the event expression result is evaluated to be non-zero. Accumulation will continue until the event expression is evaluated to give a zero result, at which point the Totalizer is closed out and placed in a non-calculating state. The Totalizer point will be marked with the system status condition “Good-Off” to indicate that no postprocessing data will be available for the interval.

The times at which the event expression is evaluated is different depending on whether or not a tag is specified in the event expression. See the description of the **EventExpr** attribute for details about when the **EventExpr** is evaluated.

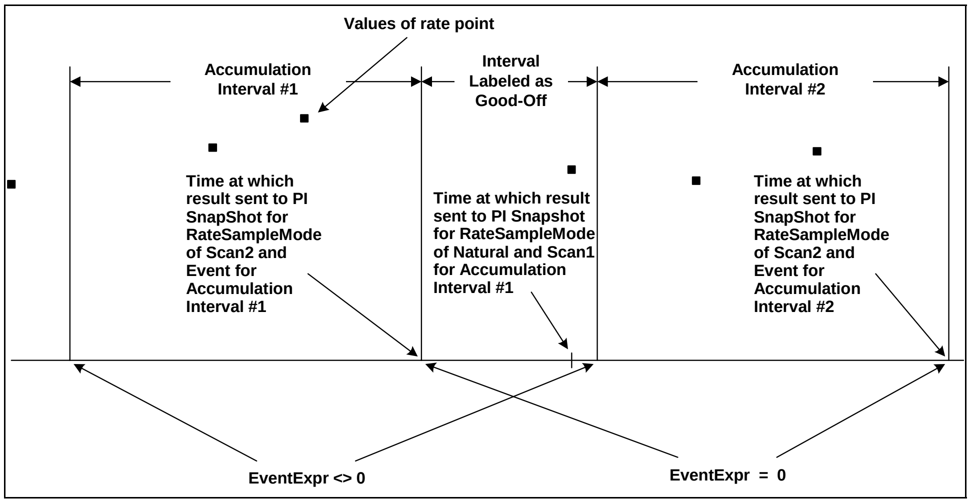

Values of rate point
Accumulation Interval #1
Interval Labeled as Good-Off
Accumulation Interval #2
Time at which result sent to PI SnapShot for RateSampleMode of Scan2 and Event for Accumulation Interval #1
Time at which result sent to PI Snapshot for RateSampleMode of Natural and Scan1 for Accumulation Interval #1
Time at which result sent to PI SnapShot for RateSampleMode of Scan2 and Event for Accumulation Interval #2
EventExpr <> 0
EventExpr = 0

Figure 7–10. TotalCloseMode of EventTrue

Similar to *EventChange*, the time at which the results are sent to PI Snapshot for **RateSampleMode** of *Scan2* and *Event* is when the event expression is equal to zero and closes out the accumulation. The time at which **RateSampleMode** of *Natural* and *Scan1* closes out the accumulation is when an update from the rate point occurs.

NSampleMoving

The accumulation interval is defined as the time required to sample the rate point a specified number of times. The actual time will be based upon the **RateSampleMode** and the filter expression. The number of values used in the calculation for this **TotalCloseMode** option is set in the **MovingCount** attribute. The *NSampleMoving* option only supports the **ReportMode** option of *Ramping*.

Figure 7–11 shows two accumulation intervals with the **MovingCount** set to **3**. Also, a filter expression is used to limit the values of the **RateSampleMode** option of *Natural* used in the accumulation.

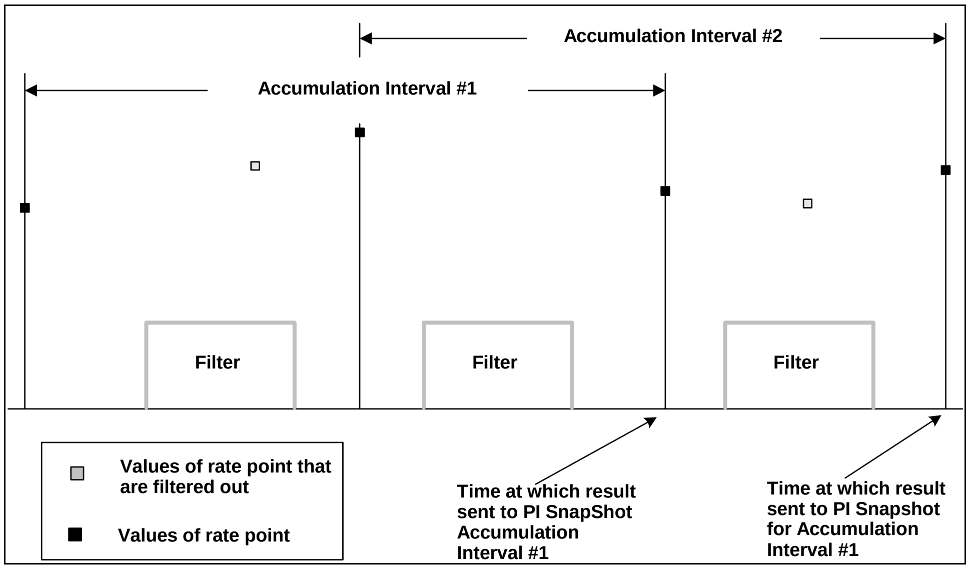

Accumulation Interval #2
Accumulation Interval #1
Filter
Filter
Filter
Values of rate point that are filtered out
Values of rate point
Time at which result sent to PI SnapShot Accumulation Interval #1
Time at which result sent to PI Snapshot for Accumulation Interval #1

Figure 7–11. TotalCloseMode of NsampleMoving (1)

The accumulation waits for the number of inputs defined by **MovingCount** and then performs the calculation and sends the results to PI Snapshot. As a new sample arrives, the value at the beginning of the accumulation is left out to incorporate the most recent value and a new result is calculated and posted. In the above example, the values that are used do not include the values excluded by the filter expression. For the **RateSampleMode** of *Scan1*, *Scan2*, and *Event*, values that are included in the accumulation are interpolated from the values of the rate point.

Figure 7–12 shows an example of **RateSampleMode** of *Scan1* with a filter expression.

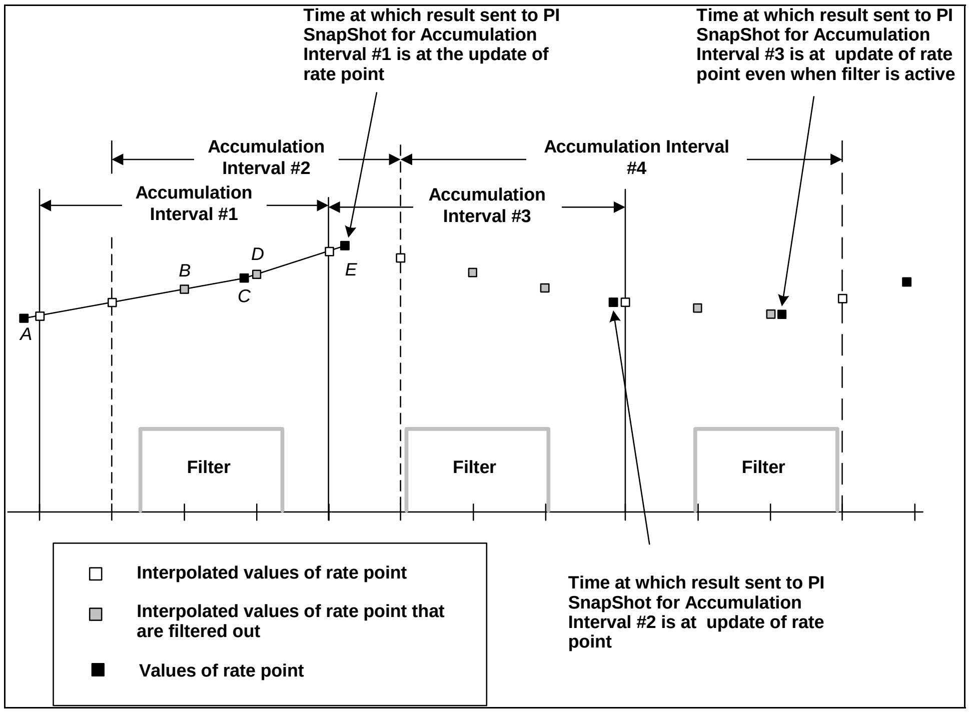

Time at which result sent to PI SnapShot for Accumulation Interval #1 is at the update of rate point
Time at which result sent to PI SnapShot for Accumulation Interval #3 is at update of rate point even when filter is active
Accumulation Interval #2
Accumulation Interval #4
Accumulation Interval #1
Accumulation Interval #3
A
B
C
D
E
Filter
Filter
Filter
Interpolated values of rate point
Interpolated values of rate point that are filtered out
Values of rate point
Time at which result sent to PI SnapShot for Accumulation Interval #2 is at update of rate point

Figure 7–12. TotalCloseMode of NSampleMoving (1)

Due to the configuration of the scan rate, the number of accumulations for the same number of updates for the rate point doubles to four intervals. In the figure, the update values from the rate points labeled as A and C are used to generate three interpolated values. Although the update of the rate point C is within the time when the filter expression is true, it is used as a reference point to generate interpolated values from update rate point values labeled as A and C and also from C to E. Notice that the interpolated values (labeled as B and D) that occurred when the filter expression was used, were excluded. Furthermore, in this example, the times at which the results are sent to the Snapshot are at updates of the rate point.

NSampleBlock

The accumulation interval is defined as the time required to sample the rate point a specified number of times. The actual time will be based upon the **RateSampleMode** and the filter expression. The number of values used in the calculation for this **TotalCloseMode** option is set in the **MovingCount** attribute. With *NSampleBlock*, the accumulation is in blocks of events. The *NSampleBlock* option only supports the **ReportMode** option of *PeriodEnd*.

Figure 7–13 shows three accumulation intervals with the **MovingCount** set to **3**. Also, a filter expression is used to limit the values of the **RateSampleMode** option of *Natural* used in the accumulation.

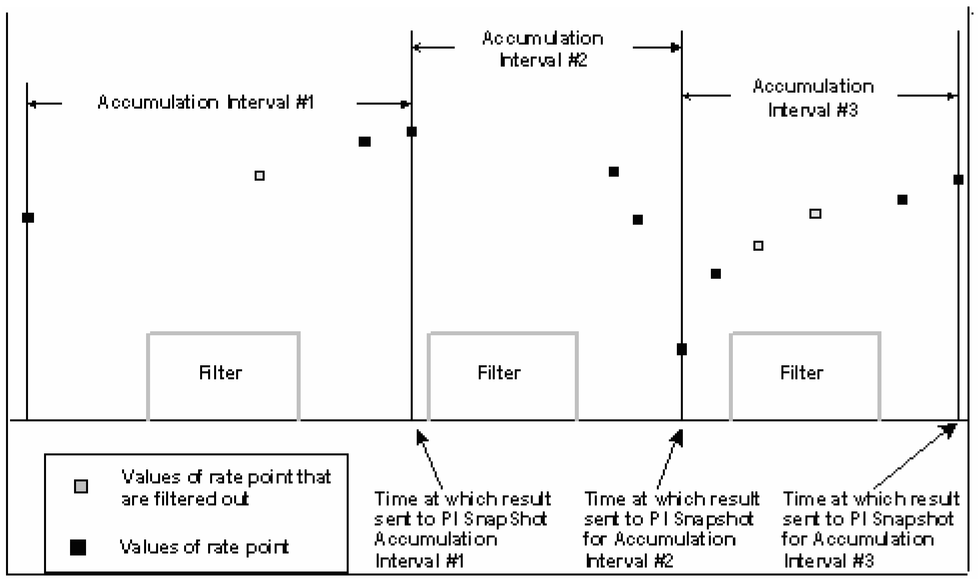

Accumulation Interval #1
Accumulation Interval #2
Accumulation Interval #3
Filter
Filter
Filter
Values of rate point that are filtered out
Values of rate point
Time at which result sent to PI SnapShot Accumulation Interval #1
Time at which result sent to PI Snapshot for Accumulation Interval #2
Time at which result sent to PI Snapshot for Accumulation Interval #3

Figure 7–13. TotalCloseMode of NSampleBlock

The accumulation waits for the number of inputs defined by **MovingCount** and then performs the calculation and sends the results to PI Snapshot. In the example (Figure 7–13), the values that are used do not include the values excluded by the filter expression. Accumulation Interval #1 shows the first accumulation interval after the start of the Totalizer, the accumulation interval starts at the time of the first event and closes at the time of the last event. The next accumulation level starts after the end of the prior accumulation interval and stops after three new events (which are not filtered) are processed. Notice that this second accumulation interval and subsequent accumulation intervals will all start at one second after the close of the prior interval.

TimeMoving

This option is similar to the previous case except the time for accumulation is specified rather than the number of samples. The accumulation time is specified in the **Period** attribute. As a value for accumulation is received, the Totalizer goes back a certain amount of time that is specified by the **Period** parameter and checks to see if the Totalizer has started or that the rate point has been created for postprocessing. If either of the conditions is not true, then the point is entered into a circular queue. When conditions (start of Totalizer and point created for rate point) are true, then the accumulation interval is set and a value for the beginning of the period is interpolated between two update values of the rate point.

Figure 7–14 demonstrates the beginning of a *TimeMoving* accumulation. In this example, the first accumulation interval does not begin until the fourth update of the rate point because the fourth point is the first instance where the accumulation interval does not extend to the start of the Totalizer.

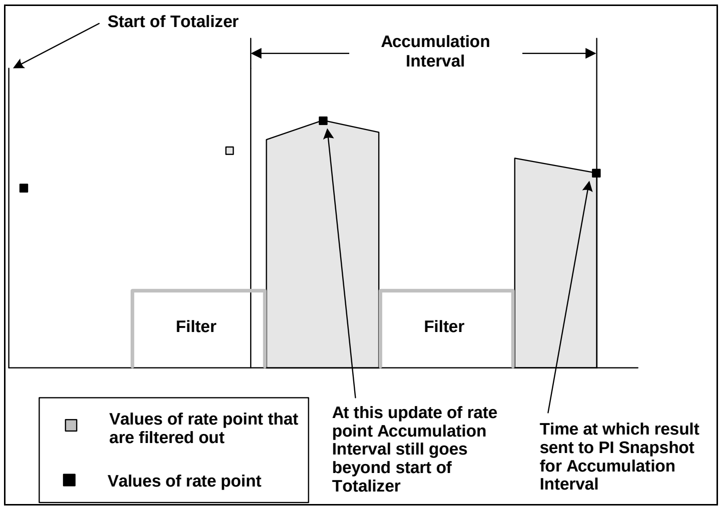

Start of Totalizer
Accumulation Interval
Filter
Filter
Values of rate point that are filtered out
Values of rate point
At this update of rate point Accumulation Interval still goes beyond start of Totalizer
Time at which result sent to PI Snapshot for Accumulation Interval

Figure 7–14. TotalCloseMode of TimeMoving and RateSampleMode of Natural

The shaded region represents the area under the curve for which postprocessing for the **CalcMode** option of *TimeWeighted* is calculated. This is discussed in **CalcMode** on page 287. In another example, the **RateSampleMode** is *Scan2* whereby the value of the last update of the rate point is used as **0the** interpolated value.

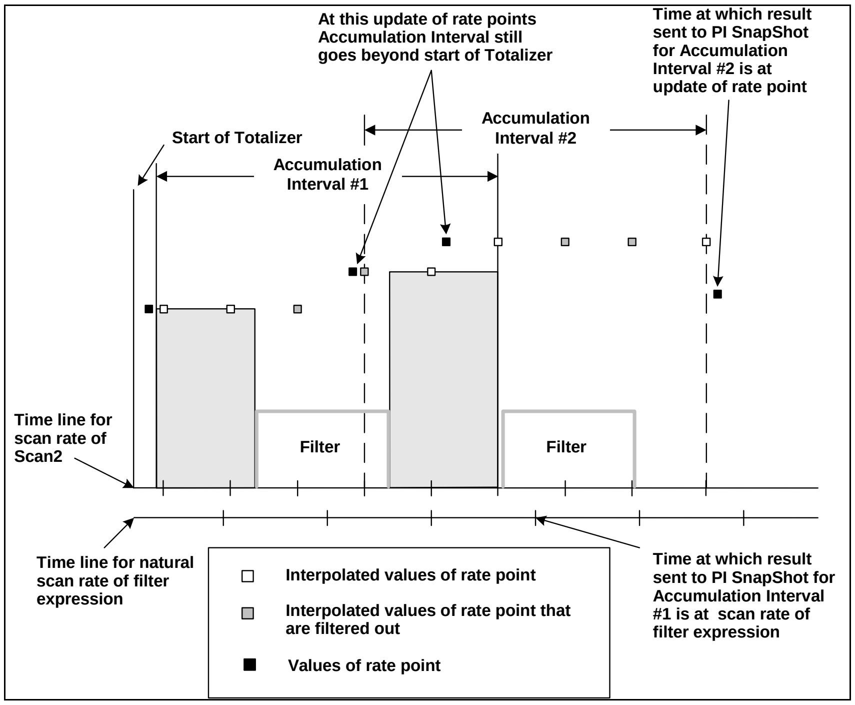

At this update of rate points Accumulation Interval still goes beyond start of Totalizer
Time at which result sent to PI SnapShot for Accumulation Interval #2 is at update of rate point
Accumulation Interval #2
Start of Totalizer
Accumulation Interval #1
Time line for scan rate of Scan2
Filter
Filter
Time line for natural scan rate of filter expression
Interpolated values of rate point
Interpolated values of rate point that are filtered out
Values of rate point
Time at which result sent to PI SnapShot for Accumulation Interval #1 is at scan rate of filter expression

Figure 7–15. TotalCloseMode of TimeMoving and RateSampleMode of Scan2

Since the interpolated values of the rate point depend only on a prior update of the rate point, the scan rate of the filter expression is also important. In the figure above, the time at which the results for accumulation interval #1 is at the natural scan rate of the filter expression and does not need to be delayed to the time of the update of the rate point. For this example, if the filter expression did not exist, then both the results from accumulation interval #1 and #2 would have been sent to the Snapshot at the update of the rate point. The start of the Totalizer is when the Totalizer Subsystem begins. If the file, *pilasttot_T.dat* does not exist in the *pi\dat* directory, the start of the Totalizer would be at the time when the Totalizer last started. If the file does exist, then the start of the Totalizer is at the time specified in the *pilasttot_T.dat.* The only valid option for the **ReportMode** attribute is *Ramping*.

Forever

In this mode, the Totalizer never resets. When restarted, it continues from the value found in the resultant point. This option works for total and all of the count and timing functions. Some external force may change the Totalizer point only if the **Option** point attribute is set to *Setable*. In this event, the new value will be used as the base for the next Totalizer result. The only valid option for **ReportMode** is *Ramping*. The only functions allowed are *Total*, *Maximum*, *Minimum* and event counting functions.

7.3.4 ReportMode

The option set in the **ReportMode** attribute specifies the manner in which the Totalizer results will be sent to the PI Snapshot. Table 7–3 lists the allowed combinations of **TotalCloseMode** and **ReportMode** where ‘X’ denotes the allowable option.

Table 7–3. Allowed Combinations of TotalCloseMode and ReportMode

	ReportMode (X indicates combinations that are allowed)				
TotalCloseMode	**Period End**	**Ramping**	**Running**	**Run Estimate**	**RunEst2**
Clock	X	X	X	X	X
EventChange	X	X	X		
EventTrue	X	X	X		
NSampleMoving		X			
NSampleBlock	X				
TimeMoving		X			
Forever		X			

PeriodEnd

No output is reported until the end of the accumulation period. At that time, the Totalizer sends the calculated result twice by default. The value is sent once with a timestamp of the start of the accumulation period plus one second, and again with a timestamp at the end of the accumulation interval. The result on a trend display in FactoryTalk Historian ProcessBook is a horizontal line of the result value through the period of the totalization. An Archive query for the result will return the same value any time during the period. The **Option** parameter may be used to change the default to write a value only at the beginning of the period (*OneAtStart*) or only at the end of the period (*OneAtEnd*).

Ramping

The *Ramping* option result is a running result that is sent to PI Snapshot when an update event occurs. Using the **Function** option *Total* with *Ramping* will result in a sawtooth trend on a trend display that ramps up and then resets.

Running (default)

The *Running* option result is output to the PI Snapshot as each rate point event is received. It is sent with a timestamp of one second into the current totalization period. If the Archive compression specification **CompMin** is set to one second, the successive values will not be recorded in the Archive, but will be available for display. Running is the default setting for **ReportMode**.

RunEstimate

A new result is sent to the PI Snapshot as each rate point event is processed. The value is an estimate of the result if the rate point were to hold steady at its current value. *RunEstimate* can only be used for the **Function** option of *Total*.

RunEst2

A new result is sent to the PI Snapshot as each rate point event is processed. The value is an estimate of the result if the rate point were to hold steady at the average observed so far in this accumulation interval. **RunEst2** can only be used for the **Function** option of *Total*.

7.3.5 Function

The combination of **Function** and **CalcMode** parameters defines the primary behavior of a Totalizer point. The first seven **Function** options (*Total*, *Average*, *Minimum*, *Maximum*, *Range*, *StdDeviation*, and *Median*) are intended for use with numerical rate points only. The first two **CalcMode** options (*TimeWeighted* and *EventWeighted*) define the kind of accumulation.

The remaining functions (*"Events:" EventEQ*, *EventNE*, …) are for counting events and are primarily intended for use with digital rate points. Besides the **Function** option of *Events*, they compare the rate point value to the **CompValue** parameter, which is expected to be the text of a digital state. They will also work with numeric points and a valid number in **CompValue**. Each is a counter for a **CalcMode** of *AllEvents* or *ChangeEvent*. The result is a time when the **CalcMode** is *TimeTrue*. Internally, the time is accumulated in seconds. The result is multiplied by the **Conversion** parameter before being sent to the PI Snapshot.

Table 7–4 shows the allowed combinations of **Function** and **CalcMode** options where **X** denotes the allowed combinations.

Table 7–4. Viable Function and CalcMode Options

	Math		**Counting**		**Timing**
Function	**Time Weighted**	**Event Weighted**	**All Events**	**Change Events**	**Time True**
Total	X	X			

	Math		**Counting**		**Timing**
Average	X	X			
Minimum	X	X			
Maximum	X	X			
Range	X	X			
StdDeviation	X	X			
Median		X			
Events			X	X	
EventEQ			X	X	X
EventNE			X	X	X
EventGE			X	X	X
EventGT			X	X	X
EventLE			X	X	X
EventLT			X	X	X
EventGE_LT			X	X	X
EventLT_GE			X	X	X

Some example combinations are as follows:

Function	**CalcMode**	**Comp Value**	**Result**
EventEQ	*ChangeEvent*	*Manual*	Number of times the rate point switched to Manual
EventNE	*TimeTrue*	*Manual*	Total time the rate point was in some state other than Manual
Events	*AllEvents*		Total number of updates received by the rate point

Total (default)

The result is the totalization of the rate point value. If **CalcMode** is *EventWeighted*, this is the sum the values from the individual rate point value update events. *TimeWeighted* uses trapezoidal integration to produce the total. Caution should be used in choosing the **RateSampleMode** option. If the goal is for a total of the values of a rate point, the *Natural* option should be used. The other **RateSampleMode** options will extrapolate values from defined time points and may lead to incorrect totals.

Average

The result is the average of the rate point value. If the **CalcMode** is *EventWeighted*, the average is the totaled individual update events and divided by the number of events. The

CalcMode option of *TimeWeighted* considers the time between events, dividing by the area under the curve by the time interval.

Maximum or Minimum

The result is the maximum or minimum observed value of the input. If the **CalcMode** is *EventWeighted*, the result will be one of the input values. If there is a filter expression active and the **CalcMode** is *TimeWeighted*, the result may be an interpolated value corresponding to the time when the filter changed state.

For **TotalCloseModes** of *NSampleMoving* and *TimeMoving*, a sorted list of all of the values seen during the period is maintained.

Range

The result is the difference between the maximum and minimum observed values of the input. If the **CalcMode** is *EventWeighted*, the result will be one of the input values. If there is a filter expression active and the **CalcMode** is *TimeWeighted*, the result may be an interpolated value corresponding to the time when the filter changed state.

For **TotalCloseModes** of *NSampleMoving* and *TimeMoving* a sorted list of all of the values seen during the period is maintained.

StdDeviation

The result is the standard deviation of the observed data. Both time- and event-weighted forms are available.

For **TotalCloseModes** of *NsampleMoving*, an array of intermediate results is kept. *StdDeviation* is not supported for **TotalCloseMode** option of *TimeMoving* with **CalcMode** option of *EventWeighted*.

Median

The *Median* value is the middle observed value of the input. If there is an even number of values in the sample, an average of the center two is selected. Using *Median* with a short time interval can be very effective as a filter for noise removal.

For **TotalCloseModes** of *NSampleMoving* and *TimeMoving* a sorted list of all of the values seen during the period is maintained.

Counting and Timing Functions

The following **Function** options are used for counting and timing operations. The allowed **CalcMode** options are *AllEvents*, *ChangeEvents*, and *TimeTrue*.

Events

This **Function** option counts all event updates of the rate point that pass exception handling and the filter expression. The allowed **CalcMode** options are *AllEvents* and *ChangeEvents*. Caution should be used if the **RateSampleMode** option is not *Natural*. Using the other three options of the **RateSampleMode** with *Events* will result in counting the interpolated values as event updates.

EventXX

The set of **Function** options denoted as *EventXX (EventEQ, EventNE, EventGE, EventGT, EventLE, and EventLT*) is used to count the number of update events that pass exception handling. The filter expression or is used to find the time that the value of the rate point meets the condition of the **Function**. As rate point events are received, they are compared to the value in **CompValue** parameter to produce a true or false result.

For **CalcMode** of *AllEvents*, true events are simply counted. *ChangeEvents* will only count those events that have values different than the immediately previous event. Consecutive events with the same value will be skipped. For the *TimeTrue* mode, the true event starts a timer and a false event stops it.

EventXX_XX

The set of **Function** options denoted as *EventXX_XX (Event LT_GE and EventGE_LT***)** is used as transition counters. They are used to count the number of times the update events (that pass exception handling and the filter expression) pass through the value set in the **CompValue** attribute or are used to find the time that the value of the rate point meets the condition of the **Function**. As rate point events are received, they are compared to the value in **CompValue** parameter to produce a true or false result.

For **CalcMode** of *AllEvents*, true events are simply counted. *ChangeEvents* will only count those events that have values different than the immediately previous event. Repeat events with the same value will be skipped. For the *TimeTrue* mode, the true event starts a timer and a false event stops it.

7.3.6 CalcMode

The value set in the **CalcMode** parameter modifies the **Function** parameter. The first two options (*TimeWeighted* and *EventWeighted*) can only be used with math functions while the last three (*AllEvents*, *EventChange*, and *TimeTrue*) can only be used with counting functions.

TimeWeighted

The time between input values is (default)considered in totalization and average functions. This is the normal option for accumulating flow signals into totals for the period. Figure 7–16 demonstrates a time-weighted total for both **RateSampleMode** options of *Natural* and *Scan1*.

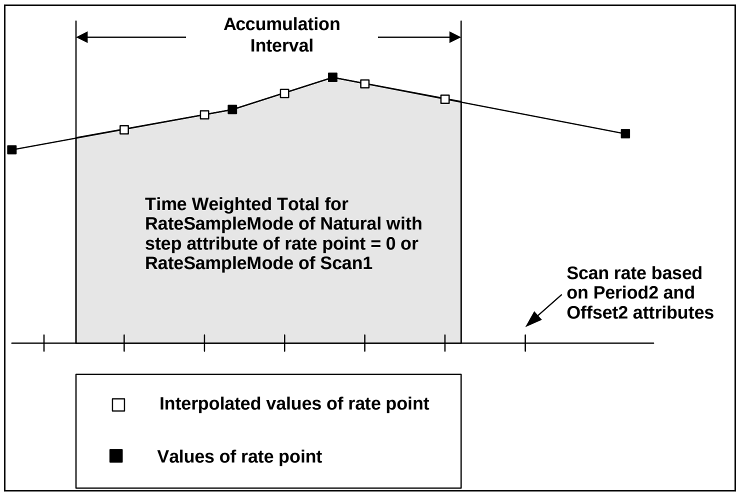


Figure 7–16. TimeWeighted Total for Natural and Scan1 with Step=0

In both cases, the total will be similar. The area considered for totalization is the shaded region. Furthermore, if the **Step** attribute of the rate point is **0**, then that indicates that the values between the update points are interpolated. If the value of the **Step** attribute is **1**, then the *Scan1* **RateSampleMode** is overwritten and the totalization would be as if *Scan2* was the intended option. Figure 7–17 shows the plots for a totalization for the **RateSampleMode** options of *Natural* and *Scan2* with the **Step** option set at **1**.

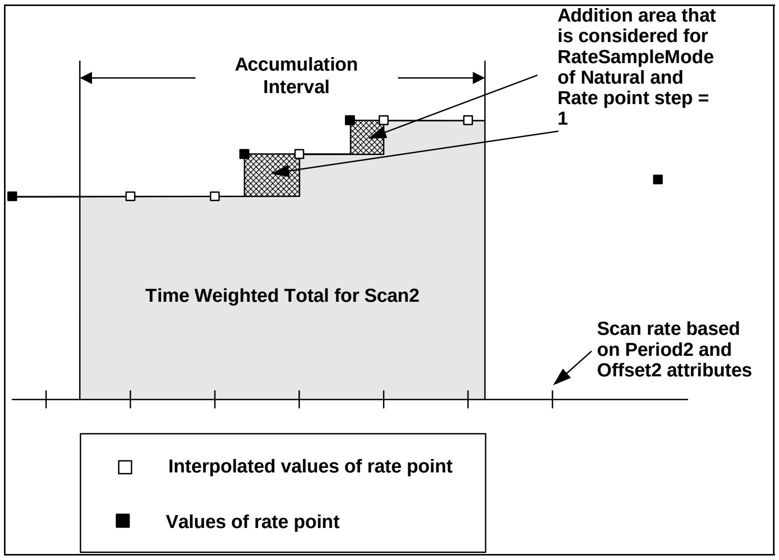


Figure 7–17. Time-Weighted Total for Natural and Scan2 with Step=1

When the **Step** attribute is “on” (equal to 1) then the value of the rate point remains the same until a new update value is seen. This is similar to *Scan2*. For the example in the previous figure, the totalization for *Scan2* is shown in the shaded area. This, however, is different than the area for *Natural* scanning with **Step** equal to **1**. For this specific example, the totalization considers both the two shaded patterned areas as well as the shaded area for *Scan2*. Even if **Step** = **0** for *Scan2*, the totalization would be the same.

EventWeighted

Here each rate point value is taken to be a discrete addition to the accumulated result. Time is not considered. Each event has the same weight. In the first example shown in Figure 7–18, the **RateSample** mode is *Natural* and the **Function** is *Total*.

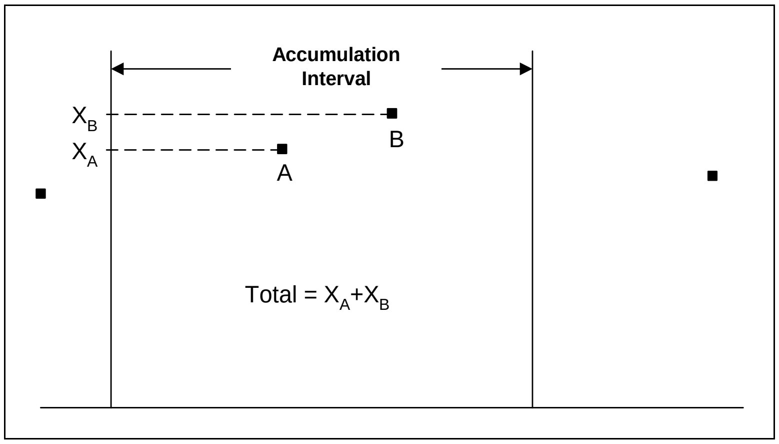


Figure 7–18. Event-Weighted Total for Natural

In this case the accumulation interval contained two updates of the rate point labeled as A and B. Therefore, the result is the summation of the values (XA and XB) of the rate points.

In another example shown in Figure 7–19 the **RateSampleMode** is *Scan1* and the **Function** is *Average*.

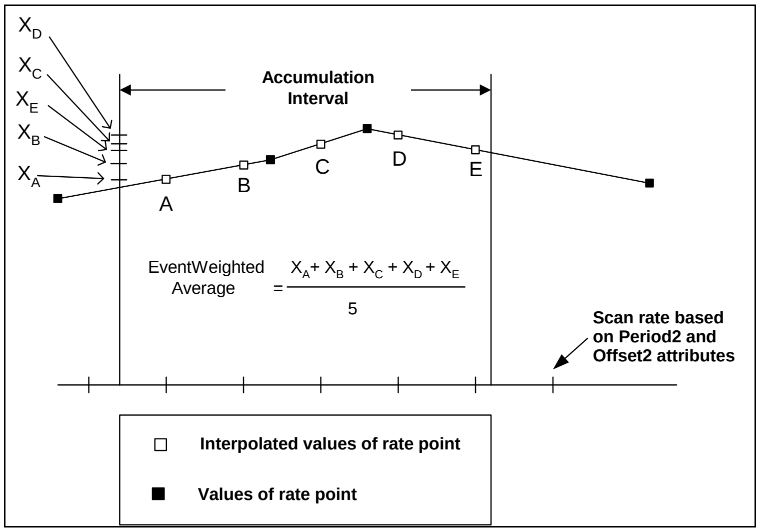

X_D
X_C
X_E
X_B
X_A
Accumulation Interval
A
B
C
D
E
EventWeighted Average = $\frac{X_A + X_B + X_C + X_D + X_E}{5}$
Scan rate based on Period2 and Offset2 attributes
□ Interpolated values of rate point
■ Values of rate point

Figure 7–19. Event-Weighted Average for Scan1

The scan rate generated five interpolated values labeled as XA through XE, and the average would be the sum of the values divided by the number of interpolated events. Use caution if you want to use the **Function** option of *Total* with *EventWeighted* and the **RateSampleMode** options of *Scan1*, *Scan2*, and *Event*. In the above example, since the events were interpolated, the total from the interval would be the summation from XA through XE.

AllEvents

With the **Function** option of *Events*, the **CalcMode** option *AllEvents* counts all value updates. For the rest of the event functions that use the **CompValue** attribute for comparison with the update value, this **CalcMode** option counts all the events for which the comparison is true. This means that events that repeat the same value will be counted.

In Figure 7–20, the digital states can be represented on a timeline by plotting the digital state offsets for the two digital states "on" and "off."

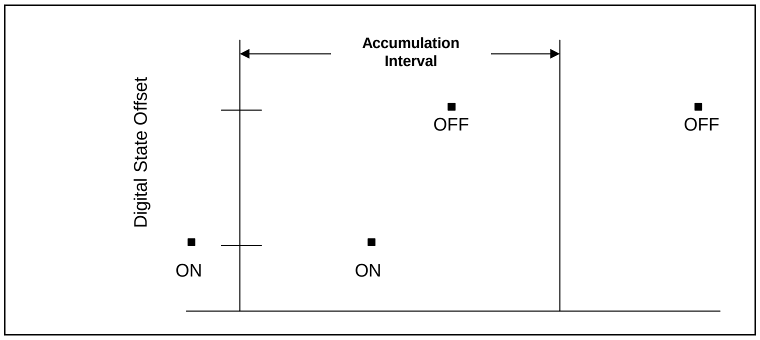


Figure 7–20. Digital States with RateSampleMode of Natural

For the **Function** of *Events*, the count for the accumulation interval is **2**. If the **Function** is *EventEQ* and the **CompValue** is "on," then the result is **1**. For functions such as *EventGT*, the comparison is made based on the digital state offset. Since "on" has a lower digital state offset value than "off" for this example, using *EventGT* would result in counting the "off" event that occurred within the accumulation interval.

In another example shown in Figure 7–21, the **RateSampleMode** is *Scan1*. The value of the digital state at the time set by the scan rate is the digital state for the last seen update event. Hence, using *Scan1* or *Scan2* would be similar.

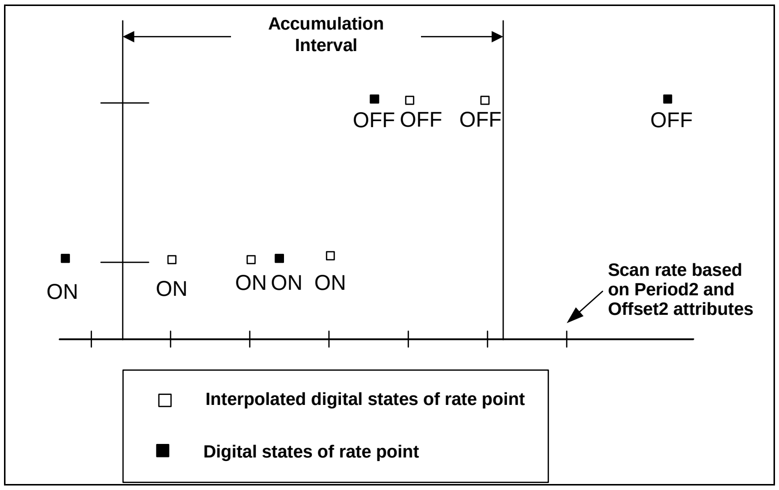

Accumulation Interval
OFF OFF OFF
OFF
ON
ON
ON ON ON
Scan rate based on Period2 and Offset2 attributes
Interpolated digital states of rate point
Digital states of rate point

Figure 7–21. Digital States with RateSampleMode of Scan1 or Scan2

The result of **Function** option *Events* is **5** and the result of *EventEQ* is **3** and the result of *EventGT* is **2**.

ChangeEvents

In the event class of functions, the count is only those updates that satisfy the condition and that were real changes of value. Therefore, the count represents not the update events but the changes of the update events. For the example shown in *AllEvents* where the **RateSampleMode** could be either *Scan1* or *Scan2*, the result of the function option *Events* is **2** and the result of *EventGT* is **1**.

TimeTrue

This **CalcMode** option results in the duration of the conditions to be accumulated. Internally, time is accumulated in seconds. The result is multiplied by the **Conversion** parameter before being sent to the PI Snapshot. For the **Function** of *EventEQ* and **CompValue** of “on”, *TimeTrue* will find the time within the accumulation interval that the digital state was “on”.

7.3.7 ZeroBias

The value of the rate point will be considered to be zero if less than this value.

7.3.8 Period

The **Period** attribute is used by the **TotalCloseMode** option of *Clock* to set the accumulation interval. It accepts relative timestamps. For example, ‘+1h’, ‘+30m’, and ‘+1d’ will set the accumulation interval to one hour, 30 minutes, and one day, respectively.

Periods longer than one hour are understood to be in local (wall clock) time. That is, for periods that evenly divide into 24 hours, the totalization times will be the same from day to day as the system changes to/from Daylight Savings Time (DST).

Optionally, the time interval may be specified to be in local (wall clock) or UTC (fixed period) terms. This is done by appending a **/local** or **/utc** to the parameter. Actually, only the **l** or **c** is needed.

For example, '`+8h /local`' specifies 8 hour shifts instead of absolute 8 hour periods. This means that for the 8-hour totalization period that spans the time change from Daylight to Standard, the actual period length would be 9 hours. When the time changes from Standard to Daylight, the actual period would be 7 hours.

7.3.9 Offset

The relative timestamp set in the **Offset** attribute is used by the **TotalCloseMode** option of *Clock* to determine when to begin the initial accumulation. The **Offset** is a relative timestamp that must be less than the relative timestamp set in the **Period** attribute. For example, if **Offset** is '`+12h`' and **Period** is '`+1d`', the Totalizer will begin the accumulation calculation at noon on the day that the Totalizer starts and will accumulate data for a period of 24 hours.

7.3.10 MovingCount

The value in **MovingCount** attribute is used by the **TotalCloseMode** option of *NSampleMoving* and *NSampleBlock* to determine the number of samples for the accumulation interval.

7.3.11 Period2

The **Period2** attribute is used by the **RateSampleMode** options of *Scan1* and *Scan2* to set the sampling rate. It accepts relative timestamps. For example, '`+1m`', '`+30s`', and '`+ 1h`' will set the accumulation interval to one minute, 30 seconds, and one hour, respectively. The **Period2** attribute also accepts the optional **/local** and **/utc** flags.

7.3.12 Offset2

The relative timestamp set in **Offset** attribute is used by the **RateSampleMode** option of *Scan1* and *Scan2* to determine when to start sampling. **Offset2** is a relative timestamp that must be less than the relative timestamp set in the **Period2** attribute. The Totalizer will use the beginning of the day for which the Totalizer is started as the reference point for when to account for **Offset2** and move in intervals set by **Period2** and sample at the next appropriate beginning of a period. For example, if **Offset2** is '`+10s`', **Period2** is '`+1m`', and the start of the Totalizer is at noon, the first sample will be taken at noon plus 10 seconds and the next sample will be taken at 12:01:10.

7.3.13 PctGood

PctGood is a number between 0 and 100 (in percent). If rate tag values are bad for a larger fraction of the totalization period, the output of the Totalizer will be marked bad.

Percent good is calculated based on the amount of time that the rate tag has a bad status over the totalization period for *TimeWeighted* totals and for the event counting functions *(Events, EventEQ, EventNE, EventGE, EventGT, EventLE, EventLT, EventGE_LT, EventLT_GE)* with a **CalcMode** of *TimeTrue*.

Percent good is calculated based on the number of events that have a bad status for *EventWeighted* totals and for the event counting functions with a **CalcMode** of *AllEvents* and *ChangeEvents*.

In the case of *TimeWeighted* totals, the Totalizer adjusts the total to account for the missing data. This extrapolation takes the resultant good data and divides it by the fraction of time that the data was good. For example, if the total of the good data is 100 and the percentage of the data that was good for the time period is 80%, then the total that is reported is 125. To turn off extrapolation, set **PctGood** to **0**. By setting **PctGood** to **0**, the total that is reported from the previous example would be 100.

7.3.14 Conversion

Conversion is a number that multiplies the raw Totalizer result. It is used to convert the units of the rate tag to the proper units for the totalization.

For *TimeWeighted* totals, the total is computed with the assumption that the rate tag is in “units/day.” If the units of the rate tag are not in units/day, then the units of the rate tag must be converted to units/day using **Conversion**. For example, if the units of the rate tag are in kg/hr (kilograms per hour) and the desired total is in **g** (grams), then **Conversion** must be set to 2400. That is, (1 kg/hr) (24 hr/day)(1000 g/kg) = (2400 g/day).

For *EventWeighted* totals **Conversion** can be used to convert the units of the rate tag to the desired units of the total. For example, if the units of the rate tag is in kilograms and the desired total is in grams, then **Conversion** should be set to 1000.

Here are some typical conversion factors for common units.

Table 7–5. Conversion Factors for Units

Source Units	Total Units	Conversion
bbl per day	bbl	1.0
lbs per hour	lbs	24.
gal per min	gal	1440
Cubic feet per sec.	acre-ft	0.504167

7.3.15 FilterExpr

This is a mathematical expression in PI Performance Equation syntax. The expression is compiled and given an initial evaluation based on the current Snapshot values of tags referenced. It is re-evaluated whenever an update is received for any of its tags.

Rate tag point values are excluded from totalization during intervals that the expression result is zero. That is, rate tag values for periods when the filter expression result is zero are not included in the total. This behavior is consistent with the sense of filter expressions in PI-API and FactoryTalk Historian DataLink. For averages, the time is also excluded from the calculation.

For efficiency of evaluation, Archive access functions are not allowed in filter expressions. Archive access functions include such functions as **TagVal** and **TagAve**.

The rate point is not filtered when the status of the filter expression is bad. The filter expression is likely to be bad whenever some of the input points for the filter expression have bad status.

Refer to Chapter 2, *PI Performance* Equation, for more information about the syntax of the expressions.

Some examples of filter expressions are as follows:

```
'DigitalTag'= "ON"
'RateTag' > 50
```

7.3.16 EventExpr

This is a PE expression that defines event times used by **EventChange** and **EventTrue** for the **TotalCloseMode** options. It also defines the times at which the rate point is scanned for the **RateSampleMode** option of **Event**. The expression is naturally scheduled. That is, it will be evaluated whenever a new value is received for any of its input values.

Event expressions are normally evaluated as new values are received for the tags they reference. For **TotalCloseMode** of **EventChange** and **EventTrue**, an event expression that references no Snapshot tags will be evaluated at times specified by the **Period** and **Offset** parameters. Each time a rate tag value is received, if the event expression due time has passed, the rate tag value is evaluated prior to processing the new update value.

Some examples of event expressions are as follows:

```
'R-2410_mode'
'DigitalTag' = "ON"
int( ('*' - '1-jan-2001')/(7 * '+24h') )
```

7.3.17 CompValue

The value set in the **CompValue** attribute is used for comparison in all the event counting functions (e.g., *EventGE*, *EventLE*) except the **Function** option *Events* because *Events* counts all update events and does not need a comparison value. The **CompValue** should match the data type of the rate tag. It may be a digital state name, string, or a number.

7.3.18 Options

This field allows for entry of zero or more *option* words to select lesser-used functions.

InFromTotalizer

This option is set if the rate point is actually the integral of the flow signal to be totalized. This is intended for use with the typical DCS accumulator block. The DCS can easily process the flow at a very high sample rate. The updates to PI can be set for relatively large exception deviation values with negligible loss of accuracy. These DCS blocks typically have a rollover at some finite value. As that total is reached, the block output drops to the bottom of its range and accumulation continues. The Totalizer observes this large value jump and calculates the correct increment from the configured full-scale value of the rate point.

UnderIsBad

Consider values below the rate point zero level to be bad. If this option is not set, values are considered valid. Negative values are set to zero or used if enabled by the *NoClampZero* option.

NoClampZero

Do not clamp the output of the Totalizer to be greater than zero. Without this option, zero is substituted for negative rate tag values.

CloseAtEvent

The close out of the accumulation interval is at the end of the period and the result of the calculation is sent to the Snapshot. For time-weighted calculations, the value used at the end of the period is the value of the last seen update of the rate point. Figure 7–22 shows an example of a time-weighted total with *Scan1* and the *CloseAtEvent* option.

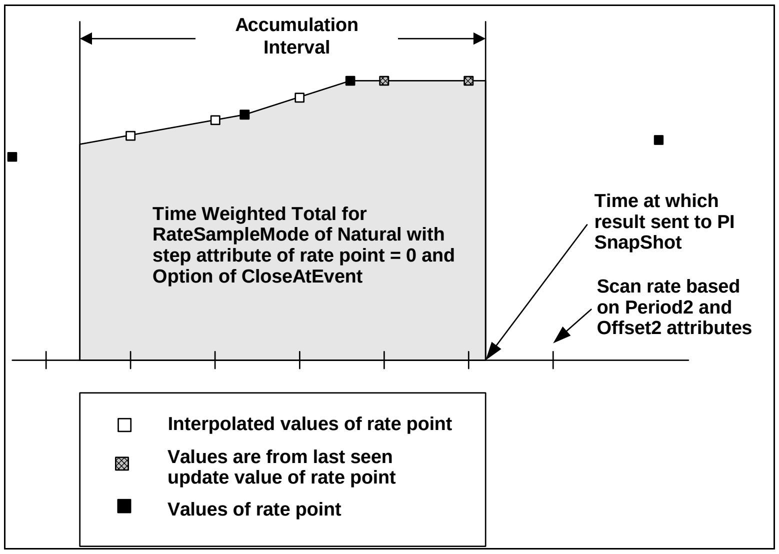

Accumulation Interval
Time Weighted Total for RateSampleMode of Natural with step attribute of rate point = 0 and Option of CloseAtEvent
Time at which result sent to PI SnapShot
Scan rate based on Period2 and Offset2 attributes
□ Interpolated values of rate point
▨ Values are from last seen update value of rate point
■ Values of rate point

Figure 7–22. Time-Weighted Total with Scan1 and CloseAtEvent

This option will works only on event-driven **TotalCloseModes** of *EventChange* and *EventTrue*.

Setable

The **TotalCloseMode** of *Forever* can be changed by some external force only when the option field is set to *Setable*. In this case, the new value will be used as the base for the next Totalizer result.

OneAtStart

The **TotalCloseMode** of *PeriodEnd* can be modified to only send a value only at the beginning of the period (one second after the close of the last period) when the option field is set to *OneAtStart*. The default behavior of *PeriodEnd* is to send a value to the beginning and the end of the period.

OneAtEnd

The **TotalCloseMode** of *PeriodEnd* can be modified to only send a value at the end of the period when the option field is set to *OneAtEnd*. The default behavior of *PeriodEnd* is to send a value to the beginning and the end of the period.

UnderIsZero

Some external systems have limited ranges for measured values but are able to report that the signal is below the available range. This option causes the Totalizer to use the rate tag zero value in place of updates with **UnderRange** status.

OverIsTop

Some external systems have limited ranges for measured values but are able to report that the signal is above the available range. This option causes the Totalizer to use the rate tag top of span value in place of updates with **OverRange** status.

SourceStat

When the **PctGood** minimum is not attained, the Totalizer normally reports “bad total” as a status. With this option, a bad status actually received from the source tag will be used instead. This supplies some reason for the failure to someone looking only at the Totalizer results.

LimitBack

When *LimitBack* is set, the number of totalization periods that will be reported between source tag events is limited to twenty. This is a partial solution to the problem some systems encounter when restarting after being down for some time. The Totalizer may attempt to write results for all of the totalization periods for the period when the system was down.

7.4 Build Totalizer Points

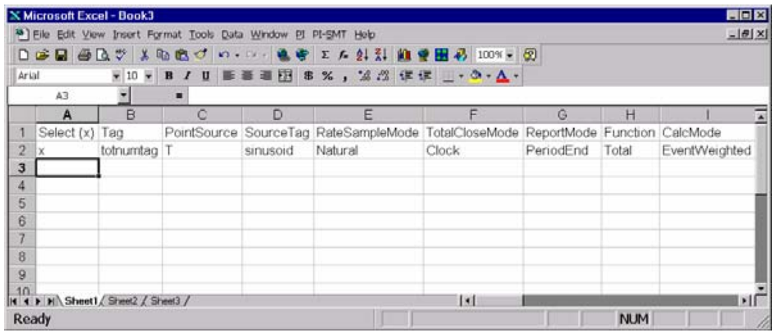
Totalizer points can be created in the same manner as other point types except that Totalizer point attributes includes the Totalizer attributes as well as the point attribute for the Base point class.

7.4.1 SMT Totalizer Editor Plug-in

This plug-in provides a user-friendly GUI to create a Totalizer point. Unlike **TagConfigurator** discussed below, the editor strictly enforces the non-supported combinations of options.

7.4.2 PI TagConfigurator

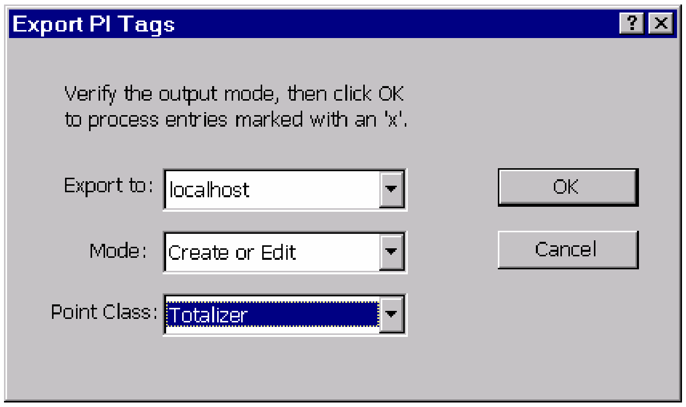
The easiest way to build many Totalizer points is using the PI **TagConfigurator** tool. See the Rockwell Automation support Web site to download this tool. An example is shown in Figure 7–23.



Select (x)	Tag	PointSource	SourceTag	RateSampleMode	TotalCloseMode	ReportMode	Function	CalcMode
x	totnumtag	T	sinusoid	Natural	Clock	PeriodEnd	Total	EventWeighted

Figure 7–23. PI TagConfigurator Creation of Totalizer Point

Note: under **Export Tags** of the **PI SMT** pull-down menu, the Point Class must be **Totalizer**.



Export PI Tags

Verify the output mode, then click OK
to process entries marked with an 'x'.

Export to: localhost

Mode: Create or Edit

Point Class: Totalizer

Figure 7–24. Export Tags Dialog Box

Furthermore, the **PointSource** attribute must also be explicitly set to **T**.

7.4.3 Piconfig

Alternatively, you can use the following script in the **piconfig** utility to create the same Totalizer point:

```
@table pipoint
@ptclass totalizer
@mode create, t
@istru tag, pointsource, sourcetag, ratesamplemode, totalclosemode, reportmode,
function, calcmode
```

```
totnumtag, T, sinusoid, natural, clock, periodend, total, eventweighted
@ends
```

7.5 Program Operation

7.5.1 Startup

The **pitotal** program is started with the rest of the system. The program periodically writes its internal status to the file *PI\dat\pilasttot_T.dat*. This file is also written during a normal system shutdown. At startup, **pitotal** looks for this file. If it is not found, all Totalizer points are initialized to start at the current time without consideration of any prior history.

If the *pilasttot_T.dat* file is found, it is read to obtain partial accumulation results to include when restarting points. For the most part, all points with simple time-based scheduling will be able to restart. If the accumulation interval that was in progress when the file was written has expired, an attempt is made to close that total and a shutdown event will be written after it. Otherwise, the accumulation is restarted with the assumption that no data points have been lost. If any point appears to have shutdown events after the time of the *pilasttot_T file*, then restart is canceled in favor of total cold start. Points that are event-scheduled have no basis to know if restart is valid. Shutdown events are written and the point is cold-started.

Any point that has been reconfigured while **pitotal** was offline would be cold-started. Point Database edits may result in a re-initialization of the point. This will occur when a significant parameter change is made.

7.5.2 Error Messages

All significant events in the life of a Totalizer point are noted in the PI server log. These include point additions and edits, scan status changes, and error conditions. These events can be monitored using the **pigetmsg** in the *pi\adm* directory. For UNIX, an additional log file *pitotal.log* is also used. This file is located in the *pi/log* directory. See Chapter 2 in the ***PI Server System Management Guide***, for information about using the **pigetmsg** utility.

7.5.3 Response to Scan Flag

When an operating Totalizer point is turned off-scan, the Totalizer is closed out as though this were the end of the period and a **Scan Off** event is written to the Archive. Going back on-scan is a full initialization of the point; no partial results are carried forward. A **Scan On** event is written to the Archive.

7.6 PI for OpenVMS Upgrade Considerations

The Totalizer can nominally do all the operations that the PI for OpenVMS version could do and more. Simple time-based points should port with no difficulty. However, the advanced Totalizer algorithms depend on very specific behavior of underlying implementations that are quite different. Any PI for OpenVMS points that use filter and event expressions will need special attention.

The following are known differences:

- ❑ The meaning of the filter expression result has been reversed to match the logic of the API. Now, input values are ignored (filtered out) when the filter expression result is zero.
- ❑ The Performance Equation syntax used in the event and filter expressions is different. The largest difference is the absence of a formula library in PI Server. See Chapter 2, *PI Performance* Equation, for further details.
- ❑ New functions are available for counting and timing of events. Use of these functions may enable the elimination of many event and filter expressions.
- ❑ The use of PI for OpenVMS option of moving Totalization with event- and clock-scheduling is not supported.

7.6.1 Features in PI3 versus PI for OpenVMS

The Totalizer Subsystem includes new features that not included in PI for OpenVMS.

- ❑ Totalizer points use a new point class with a new set of parameter names and meanings.
- ❑ A running account of the output.
- ❑ The event expression that can be time-scheduled.
- ❑ New event functions that can count and time values from Digital points.
- ❑ Sampling of the rate point that can be time-scheduled.
- ❑ Moving totals with several new scheduling options.
- ❑ New numeric functions that include *Maximum*, *Minimum*, *Median*, and *Range*.
- ❑ The input that may come from a typical DCS Totalizer block.

7.6.2 Compatibility with PI for OpenVMS

The PI Server Totalizer can do all of the functions offered by PI for OpenVMS postprocessing. Simple Totalizers can be ported easily through a translation of parameter names. Points that use event or filter expressions will need to be examined for compatibility with the PI Server syntax.

In many cases it should be possible to get the desired functionality by using the new Totalizer features rather than by re-writing expressions. These will be easier to document and use less computing resource.

The meaning of the filter operation has been redefined. A true filter result, that is non-zero, now passes the signal. Table 7–6 lists the available postprocessing options in PI for OpenVMS and the PI Server equivalents.

Table 7–6. PI for OpenVMS and PI for NT and UNIX Equivalents

PI for OpenVMS		**PI Server**	
Postprocessing Type	**Type of Scheduling**	**Point Attribute**	**Option to Set**
Time-weighted	Clock	RateSampleMode	*Natural*
		TotalCloseMode	*Clock*
		ReportMode	*PeriodEnd*
		CalcMode	*TimeWeighted*
	Event	RateSampleMode	*Natural*
		TotalCloseMode	*EventChange*
		ReportMode	*PeriodEnd*
		CalcMode	*TimeWeighted*
Discrete	Clock	RateSampleMode	*Natural*
		TotalCloseMode	*Clock*
		ReportMode	*PeriodEnd*
		CalcMode	*EventWeighted*
	Event	RateSampleMode	*Natural*
		TotalCloseMode	*EventChange*
		ReportMode	*PeriodEnd*
		CalcMode	*EventWeighted*
Moving	Natural	RateSampleMode	*Natural*
		TotalCloseMode	*NSampleMoving*
		ReportMode	*Ramping*
		CalcMode	*TimeWeighted*
	Clock	*Not Supported*	
	Event	*Not Supported*	

7.7 Demonstration Points

Demonstration points for the Totalizer subsystem are available in the *pi\adm* directory in the *totalizerpts.dif* file but are not automatically created when the system is installed. These Totalizer demonstration points can be created by redirecting the file to the **piconfig** utility. See Chapter 11, *The Piconfig Utility*, in the ***PI Server System Management Guide*** for information about using the **piconfig** utility.

```
$ piconfig < totalizerpts.dif
```

Chapter 8. PI ALARM SUBSYSTEM

The **PI Alarm Subsystem (PI Alarm)** provides the capability to establish alarms for PI points. PI Alarm allows you to track, manage and acknowledge alarm conditions caused by processes that exceed user-specified parameters.

PI Alarm can monitor many variables such as temperatures, volumes, flow rates, product quality or raw material consumption. Alarms can be triggered by the duration of an event or deviation from norm.

PI Alarm keeps a constant eye on process conditions. PI Alarm will assess the condition as well as the priority of an event, as you define it. Depending on the longevity and/or severity of the event, it can notify specific personnel. PI Alarm includes client functionality through the PI-API to alert operators to selected alarms.

Data from PI Alarm are displayed in its companion client application, PI AlarmView. Alarm conditions are historized together with an acknowledgement status. When Real-Time SQC perceives an unacceptable deviation in the process, PI SQC Alarms alert the appropriate personnel.

This chapter includes the following topics:

8.1 Alarm Subsytem Overview

A PI System often brings together information from several sources and can perform calculations that are not easily done elsewhere. Some sites may have alarm philosophies that enable them to take advantage of the PI System to provide alerts on these higher level functions.

PI Alarm provides the basic server-side functions of an alarm system. The alarm package includes the following features:

- Current and archived *Alarm States*.
- *Alarm Groups* to organize and manage alarms
- A simple *alarm detection* program for monitoring numeric, digital, and string points.
- *Alarm client* functionality available through the PI-API to alert operators to selected alarms.

The alarm package is organized into two categories.

- ❑ The first part is the **Alarm Point**. Alarms are displayed and archived as digital points. A monitoring program observes updates to numeric, digital, and string points and then tests each for configured alarm conditions.
- ❑ The second part is the **Alarm Group**. A set of alarm points can be organized into Alarm Groups. Statistics such as the number of alarm points and the number of unacknowledged alarms can be obtained for each Alarm Group. Groups can be members of other groups to form alarm hierarchies.

8.1.1 Alarm Points

An alarm point is a digital point that indicates the alarm status of a point in the PI System. PI Alarm can monitor the values of other points and set the values of alarm points when defined boundaries are exceeded. Numeric values exceeding limits or digital values changing to a special state are two such boundaries.

The digital state of an alarm point may include both the *priority* and *acknowledgement* status of the *alarm condition*.

- ❑ The **condition** of an alarm is the type of limit that the alarm is triggering. Some typical names of conditions are “high” and “low”. For example, if a numeric point is greater than a certain numeric constant, the condition of the alarm can be high.
- ❑ The **acknowledgement** status may be determined by whether a user has acknowledged the alarm, not acknowledged the alarm, or not acknowledged the alarm condition that no longer exists.
- ❑ The **priority** is level of importance given to the particular alarm.

Each alarm point has a source point, which is the tag whose value is being monitored for special changes. Figure 8–1 shows the flow diagram of an alarm point.

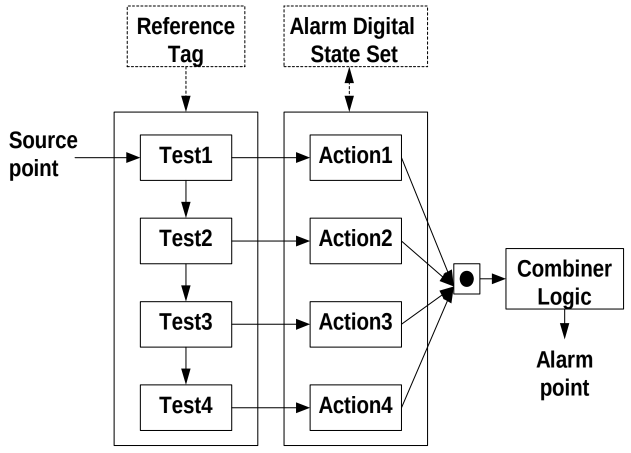


Figure 8–1. Flow Diagram of Alarm Points

The alarm point is naturally scheduled and signs up for updates of the source point. This means that the alarm point is evaluated whenever the source point produces an exception. A series of tests are done and an alarm is triggered if any of the tests are true. The digital state of the alarm point is then set based on the combiner logic, which is discussed later in this chapter.

Alarm Tests and Actions

An alarm is the result of a set of four tests on the current value of another point. Each alarm tag has a primary source point that is the subject of these tests. Each alarm point can be configured to perform four simple tests on new values received by the source tag. The tests are individually configured for comparisons like "greater than," "equal," etc. Each time a source tag value is received, the configured tests are performed. The result of each test is a Boolean (true/false) value. The four test results are input to combiner logic that decides the alarm status to be set. The answer depends on the configured priority of the test conditions, the order of detection, and the acknowledgement status of the current alarm.

Combiner Logic

Combiner logic refers to the manner in which the digital state of an alarm is set when a test or multiple tests are true. If more than one test condition is true, the rule for which Alarm State to set is that priorities have precedence. When two conditions of the same priority are true, it is the first (in order of the tests) that is used to set the Alarm State. Further details of the logic are given in the **Action1**, **Action2**, **Action3**, and **Action4** attributes for alarm points.

Acknowledgment

In this release, acknowledgment is accomplished by allowing the client program to write acknowledged digital states corresponding to the Alarm Digital States directly to the alarm points. For example, an Alarm Digital State that represents a high value in the source tag can only be acknowledged by the corresponding high acknowledged digital state. The alarm program will over-write the client input in case of an error.

See Section 8.3, *Alarm State Sets*, for more information about Alarm Digital States. Future releases will provide an interface for acknowledgement and other functions through the PI-SDK.

An auto-acknowledge for the alarm point is also possible. An auto-acknowledged point will continue to display the current alarm condition, but the display status will never be unacknowledged. Further details on auto-acknowledgement can be found in the **AutoAck** attribute for alarm points.

8.2 Alarm Point Configuration

Alarm points have the point class of **alarm**. The point source is '**@**' and the data type must be digital.

Table 8–1 lists the attributes for the alarm point class.

Table 8–1. Alarm Point Class Attributes

Point Attribute	Valid Options	Description
SourceTag		Source point used to test for alarm
Test1 **Test2** **Test3** **Test4**	*GT*	Alarm if values greater than numeric value
	LT	Alarm if values less than numeric value
	EQ	Alarm if values equal numeric value, digital state or string
	NE	Alarm if values not equal numeric value, digital state or string
	StepGT	Alarm if values stepped up more than numeric value
	StepLT	Alarm if values stepped down more than numeric value
	RateGT	Alarm if rate of change is greater than numeric value
	RateLT	Alarm if rate of change is less than numeric value
	Is_in	Alarm if value is in the digital state or string
	Not_In	Alarm if value is not in the digital state or string
	Includes	Alarm if values includes digital state or string
	Change	Alarm if value of point changes
	CondEQ	Alarm if condition is equal to string
	CondNE	Alarm if condition is equal to string
	IsUnAck	Alarm if unacknowledged
Action1 **Action2** **Action3** **Action4**	*Condition nn*	Name of the alarm condition nn = priority
	StateName	Valid name in Digital State Set.
ExDesc	*GroupTag*	Alarm Group of alarm point

Point Attribute	Valid Options	Description
DigitalSet		Name of the Alarm Digital State Set for point
ReferenceTag	*Tagname*	Point that will be used for comparison operators if specified.
AutoAck	*Yes (default)*	Any alarm always shows as acknowledged.
	No	Full acknowledge operation supported.
DeadBand	*0.0 (default)*	Threshold for point not to be in alarm
Options	*RT = num*	Used with the RateGT and RateLT to define the period of the rate
ControlTag		Point that is used to take the alarm out of service if specified.
ControlAlg		<reserved for future use>

Note: While the actual point class has the parameters, **Test5** and **Action5**, only four Test/Action pairs are implemented.

8.2.1 SourceTag

The point set in the **SourceTag** attribute is the primary source used in the testing for alarm conditions. The point must already exist and must be specified when the Alarm Point is configured. The point may be numeric, digital or string point types.

8.2.2 Test1, Test2, Test3, Test4

The set of four attributes **Test1**, **Test2**, **Test3**, and **Test4** contains the test for the alarm condition. The syntax for the attribute is the operator followed by an argument that is within parenthesis, which is also followed by an optional time parameter.

```
Syntax: Operator ( Argument ) Time
```

The argument can be a numeric constant; string, digital state, or it may refer to another tag. The following examples use the **EQ** (equals to) operator that is detailed later in this section.

```
EQ ( 12 )
EQ ( ON )
EQ ( “This is a string” )
```

In addition, the argument can be a reference to another point that has the data type of numeric, string, or digital. If a reference is used, the argument is the tagname of the reference point that is enclosed in single quotes or the keyword, *ref*. In the case where the keyword, *ref* is used, the **ReferenceTag** attribute must be the tagname of the desired reference point. An optional numeric constant offset can also be used only in the case when the keyword *ref* is used and only if the reference point is a numeric point. This adds or subtracts a numeric constant from the value of the reference point before the test is done.

```
EQ ( ‘tagname’ )
EQ ( ref )
EQ ( ref + 12 )
EQ ( ref - 14 )
```

An optional *time parameter* can also be used to delay the triggering of the alarm until the test comparison is true for a given amount of time. The time parameter is a relative timestamp that is added to the comparison. The time parameter consists of a plus sign, one or more numbers, and a letter (s, m, h, or d). There may be no spaces between these elements.

```
EQ ( 12 ) +14m
EQ (  'tagname'  ) +11h
EQ ( ref + 70 ) +1h
EQ ( ON ) +1d
```

In the examples above, the first test triggers an alarm if the source tag is equal to 12 for over 14 minutes. The second test alarms when the source point is equal to the reference point (referenced by tagname) for over 11 hours. The third test compares the source point with the reference point plus 70 and if they are equal for over one hour then an alarm is triggered. In the fourth test, an alarm triggers if the source point is **ON** for over one day.

All numeric point comparisons are performed using floating-point operations and all string and digital state comparisons are performed using character operations. Comparisons of a digital or string data type with a numeric data type returns an error at the time which the point is compiled. Blob data types are not supported. Table 8–2 shows some examples of the comparisons and whether an alarm will trigger.

Table 8–2. Example Test Comparisons and the Alarm Status

Operator	Source point value	Test value	Trigger an alarm?	Comparison
EQ	12	12	Yes	numeric source and numeric test value
EQ	14	70	No	numeric source and numeric test value
EQ	12	12	Error	numeric source and digital test value
EQ	12	"12"	Error	numeric source and string test value
EQ	OFF	OFF	Yes	digital source and digital test value
EQ	OFF	"OFF"	Yes	digital source and string test value
EQ	OFF	"OFF "	No	digital source and string test value (with trailing space)
EQ	Off	OFF	Yes	digital source and digital test value

Each of the four tests is evaluated in numeric order from **Test1** through **Test4** and each test is associated with an "action" attribute numerically. For example, if the test set in **Test1** becomes true, the associated "action" would be the one set in the **Action1** attribute. In general, if there is only one test that is used for the alarm point, it would be set in **Test1** but there is no rule governing the where to apply the test for the alarm point. **Test1** could be left unused and **Test2**, **Test3**, or **Test4** may be used. In the case where all four attributes are unused, the alarm point will default to the digital state that represents "no alarm" for that point. If more than one test is true, the digital state of the alarm is governed by the combiner logic rules mentioned earlier in this chapter and detailed in the **Action1**, **Action2**, **Action3**, and **Action4** attributes section of this chapter.

Table 8–3 shows the available options for the comparison operators with their descriptions as well as their viable argument data types. An **X** in the table denotes that the data type is allowable.

Table 8–3. Viable Argument Data Types for Operators

Operator	**Description**	**Argument Data Type (X indicates Allowed)**		
		Numeric	**Digital**	**String**
GT	Greater Than	X		
LT	Less Than	X		
EQ	Equal To	X	X	X
NE	Not Equal To	X	X	X
StepGT	Step Greater Than	X		
StepLT	Step Less Than	X		
RateGT	Rate Greater Than	X		
RateLT	Rate Less Than	X		
Is_In	Is In the string or digital state		X	X
Not_In	Is not in the string or digital state		X	X
Includes	Includes the string or digital state		X	X
Change	Change of State	X	X	X
CondEQ	Alarm condition equals		X	
CondNE	Alarm condition not equal		X	
IsUnack	Alarm condition is unacknowledged		X	

GT

The operator **GT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the value of the source point is greater than the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
GT ( 12 )
GT ( 'tagname' )
GT ( ref )
GT ( ref + 14 )
GT ( 70 ) +1h
```

LT

The operator **LT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the value of the source point is less than the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
LT ( 12 )
LT ( ‘tagname’ ) +14m
LT ( ref - 70 )
LT ( 11 ) +1h
```

EQ

The operator **EQ** tests can be performed with numeric constants, digital states, strings or a reference to another point. An alarm is triggered if the value of the source point is equal to the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point.

A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
EQ( 12.14 )
EQ ( OFF )
EQ ( ref )
EQ ( ref + 70 )
EQ ( “This is a string” ) +1h
```

NE

The operator **NE** tests can be performed with numeric constants, digital states, strings or a reference to another point. An alarm is triggered if the value of the source point is not equal to the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
NE( 12 )
NE ( ON )
NE ( ‘tagname’ )
NE ( ref - 14.11 )
NE ( “This is a string” ) +70m
```

StepGT

The operator **StepGT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the value of the source point changes more than the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
StepGT ( 12 )
StepGT ( ‘tagname’ ) +14m
StepGT ( ref - 70 )
StepGT ( 11 ) +1h
```

StepLT

The operator **StepLT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the value of the source point changes less than the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
StepLT ( 12 )
StepLT ( 'tagname' )
StepLT ( ref - 14 )
StepLT ( 70 ) +1h
```

RateGT

The operator **RateGT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the rate of change of the source point is greater than the value set in the test. The rate of change is calculated by dividing the difference between discrete average of two successive periods by the period. Figure 8–2 shows an example of the calculation of the rate of change.

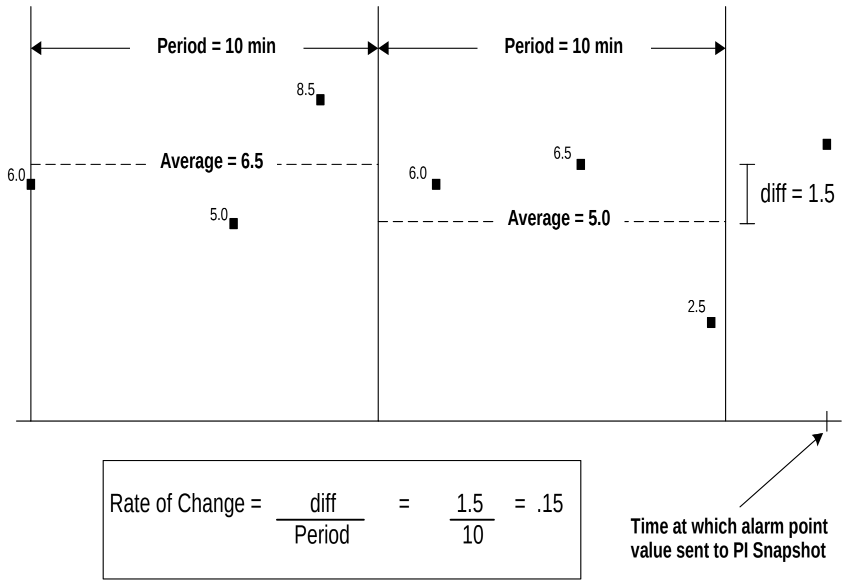

Period = 10 min
Period = 10 min
8.5
Average = 6.5
6.0
5.0
6.0
6.5
Average = 5.0
diff = 1.5
2.5
Rate of Change = diff / Period = 1.5 / 10 = .15
Time at which alarm point value sent to PI Snapshot

Figure 8–2. Calculation of Rate of Change

In the example, the average of the first period is 6.5 units and the average of the second period is 5.0 units. Hence the rate of change is 1.5 units per minute. The values used in the calculation are those sent to PI Snapshot as exceptions. The comparison test is made after the second period at the time of the arrival of the next source point Snapshot value. The default period is 10 minutes. The period can be changed by setting a new period in the options attribute.

If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
RateGT ( 12.8 )
RateGT ( 'tagname' )
RateGT ( ref + 14 )
RateGT ( 70 ) +1h
```

RateLT

The operator **RateLT** tests numeric constants as well as points that evaluate into numbers against a numeric source point. An alarm is triggered if the value of the source point is less than the value set in the test. If a reference point is used, the tagname of the reference point is used as the argument. If the keyword *ref* replaces the numeric constant as the argument and the **ReferenceTag** attribute contains the tagname of the reference point, a numeric constant may be added or subtract to the value of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time. The default period is 10 minutes. The period can be changed by setting a new period in the options attribute.

```
RateLT ( 12.8 )
RateLT ( 'tagname' )
RateLT ( ref + 14 )
RateLT ( 70 ) +1h
```

Is_In

The operator **Is_In** tests digital states, strings, or points that evaluate into digital states or strings against a digital state or string. In the comparison, all digital states are converted into strings, and a string-to-string comparison is performed. The **Is_In** operator triggers an alarm if the source point value is in the test value. For a reference point, the tagname of the reference point is used as the argument. If the keyword, *ref* is used as the argument, the **ReferenceTag** attribute contains the tagname of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
Is_In ( "ON OFF" )
Is_In ( 'tagname' )
Is_In ( ref )
Is_In ( "This is a string" ) +1h
```

Table 8–4. Comparisons of the Is_In Operator

Operator	Source point value	Test value	Trigger an alarm?	Comparison
Is_In	Off	“ON OFF”	Yes	digital source and string test value
Is_In	“string”	“This is a string”	Yes	string source and string test value
Is_In	“str”	“This is a string”	Yes	string source and string test value
Is_In	“strings”	“This is a string”	No	string source and string test value
Is_In	100	1002	Error	numeric source and digital test value
Is_In	100	“1002”	Error	numeric source and string test value

Not_In

The operator **Not_In** tests digital states, strings, or points that evaluate into digital states or strings against a digital state or string. In the comparison, all digital states are converted into strings and a string-to-string comparison is performed.

The **Change** operator triggers an alarm if the source point value is not in the test value. For a reference point, the tagname of the reference point is used as the argument. If keyword *ref* is used as the argument, the **ReferenceTag** attribute contains the tagname of the reference point.

A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
Not_In ( ″ON OFF″ )
Not_In ( 'tagname' )
Not_In ( ref )
Not_In ( ON ) +1h
```

Table 8–5. Comparisons of the Not_In Operator

Operator	Source point value	Test value	Trigger an alarm?	Comparison
Not_In	ON	“OFF ON”	No	digital source and string test value
Not_In	OF	“OFF ON”	No	digital source and string test value
Not_In	ONE	“OFF ON”	Yes	digital source and string test value
Not_In	“1”	12345678	No	string source and digital test value
Not_In	9	“12345678”	Yes	digital source and string test value
Not_In	10	100	Error	numeric source and digital test value
Not_In	10	“100”	Error	numeric source and string test value

Includes

The operator **Includes** tests digital states, strings or points that evaluate into digital states or strings against a digital state or string. In the comparison, all digital states are converted into strings and a string to string comparison is performed. The **Includes** operator triggers an alarm if the source point value includes the test value. For a reference point, the tagname of the reference point is used as the argument. If keyword *ref* is used as the argument, the **ReferenceTag** attribute contains the tagname of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
Includes ( HI )
Includes ( 'tagname' )
Includes ( ref )
Includes ( ″This is a string″ ) +1h
```

Table 8–6. Comparisons of the Includes Operator

Operator	Source point value	Test value	Trigger an alarm?	Comparison
Includes	Hihi	“HI”	Yes	digital source and string test value
Includes	High	HI	Yes	digital source and digital test value
Includes	“VIN3234A”	“VIN3234”	Yes	string source and string test value
Includes	“VIN3234A”	“VIN3235”	No	string source and string test value
Includes	1002	100	Error	numeric source and digital test value
Includes	1002	“100”	Error	numeric source and string test value

Change

The operator **Change** tests numeric values, digital states, strings or points that evaluate into numeric values, digital states or strings. All numeric comparisons are done as floating point operations and comparisons of all digital states are converted into strings and a string-to-string comparison is performed. The **Change** operator triggers an alarm if the source point value is different from the previous value. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time.

```
Change ( )
Change ( ) +1h
```

CondEQ

The operator **CondEQ** tests alarm point conditions against alarm points. An alarm will trigger if an alarm of another alarm point is equal to the test value. For a reference point, the tagname of the reference point is used as the argument. If keyword *ref* is used as the argument, the **ReferenceTag** attribute contains the tagname of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time. The argument is an alarm condition. In the following examples, an alarm is triggered if the condition of the source alarm point is **low**, the condition of the source alarm point is equal to the condition of the reference point (*ref*), and the condition of the source point is **high** for over one hour, respectively.

```
CondEQ ( low )
CondEQ ( 'alarmtagname' )
CondEQ ( ref )
CondEQ ( high ) +1h
```

Table 8–7 show examples of the **CondEQ** operator using the sample Alarm Digital State Set given in the Alarm State Sets section of this chapter. The digital state for a new alarm with the condition of low is **low <<** and the digital state for an acknowledged alarm with the condition of low and an urgent priority is **** low**.

For more information about Alarm Digital States, see Section 8.3, *Alarm State Sets*.

Table 8–7. Comparisons of the CondEQ Operator

Operator	Source point value	Test value	Trigger an alarm?	Comparison
CondEQ	Low	Low	Yes	digital source and digital test value
CondEQ	Low <<	Low	Yes	digital source and digital test value
CondEQ	** low	Low	Yes	digital source and digital test value
CondEQ	Low	Lolo	No	digital source and digital test value

CondNE

The operator **CondNE** tests alarm point against an alarm point. An alarm will trigger if an alarm of another alarm point is not equal to the test value. For a reference point, the tagname of the reference point is used as the argument. If keyword *ref* is used as the argument, the **ReferenceTag** attribute contains the tagname of the reference point. A time parameter in the form of a relative timestamp can delay the triggering of an alarm until the comparison is true for that period of time. The argument is an alarm condition. In the following examples, an alarm is triggered if the condition of the source alarm point is not **low**, the condition of the source alarm point is not equal to the condition of the reference point (*ref*), and the condition of the source point is not **high** for over one hour, respectively.

```
CondNE ( low )
CondNE ( 'alarmtagname' )
CondNE ( ref )
CondNE ( high ) + 1h
```

IsUnack

IsUnack tests an alarm point against another alarm point. This operator has no arguments and triggers an alarm if the source alarm point is unacknowledged.

In the following examples, an alarm is triggered if the source alarm point is unacknowledged and goes unacknowledged for over one hour, respectively.

```
IsUnack ( )
IsUnack ( ) +1h
```

8.2.3 Action1, Action2, Action3, Action4

The set of four attributes **Action 1**, **Action 2**, **Action 3** and **Action 4** specify the digital state that is set when the corresponding test set in the **Test1**, **Test2**, **Test3**, **Test4** attributes trigger an alarm. There are two forms of the attribute and their syntax is as follows.

```
Syntax:        Form 1) Condition Priority
               Form 2) StateName
```

If an Alarm State Set is the Digital State Set for the alarm point, then the first form of the attribute is used. In this case **Condition** is the alarm condition to be set for the alarm point and **Priority** is the numeric priority level alarm. See Section 8.3, *Alarm State Sets*, for more information about conditions and priorities. Table 8–8 gives examples of the use of the first syntax and the resulting digital state that is set using the example Alarm Digital State Set shown in Table 8–15.

Table 8–8. Examples Using First Syntax (Condition Priority)

Action1	Digital State	Description
Hihi 1	__ hihi <<	New unacknowledged HIHI alarm with priority level 1
Hihi 2	_* hihi <<	New unacknowledged HIHI alarm with priority level 2
High 3	** high <<	New unacknowledged HIGH alarm with priority level 3
Low 0	LOW	LOW alarm with priority 0 always returns just the alarm condition

In the above examples, the digital state that is set is for a new alarm with the attribute **AutoAck** set to **NO**. Furthermore, it is also possible to have a priority level of **0**, which is shown as the last example in Table 8–8. In that case, only the alarm condition is returned regardless of the acknowledgement status.

The second case is not limited to the Alarm Digital State Set and any digital state set may be utilized. The second case only requires that the attribute contain a digital state (**StateName**) that belongs to the digital state set of the alarm point.

Table 8–9. Examples Using Second Syntax (StateName)

Action 1	Digital State	Description
Hihi	HIHI	Condition only. Similar to using the first syntax with priority 0
** high	** High	Alarm Digital State
auto	Auto	Digital Set used is Modes {Manual, Auto, Cascade, Program, Prog-Auto}

The first two examples given in Table 8–9 use the Alarm Digital State Set from Table 8–15. The last example used a default digital state set, **Modes**, with the digital states {*Manual*, *Auto*, *Cascade*, *Program*, *Prog-Auto*}.

When more than one alarm is triggered simultaneously, the combiner logic determines the digital state that is set by using the following rules:

- Priorities have precedence.
- When two conditions of the same priority are true, it is the first (in order of the tests) that is used to set the Alarm State.

Table 8–10 gives examples of which "action" will be taken when more than one "action" attribute is triggered. In these examples, it is assumed that all four tests were done and the "actions" that were left blank are the result of alarms that were not triggered.

Table 8–10. Combiner Logic Examples

Action1	Action2	Action3	Action4	Result
Hihi 1		High 1		Hihi 1
Hihi 2			High 1	Hihi 2
auto	Manual			auto

In the first combiner logic example, all the priorities are the same; hence **Action1** is set because it is the first test that is true. In the second example, even though **Action4** and **Action1** are both true, **Action1** is used because it has a higher priority. The last example demonstrates the second syntax where priority is not used; hence **Action1** is set because it is the first test that is true. Refer to *Alarm State Sets* on page 321 for more information about the digital state that is set.

8.2.4 ExDesc

The tagname of the Alarm Group for the alarm point is set in the **ExDesc**. An alarm point must belong to an Alarm Group and the Alarm Group must be created before the creation of the alarm point. Refer to *Alarm Groups* on page 326 for additional information.

8.2.5 DigitalSet

The **DigitalSet** attribute specifies the name of the Alarm State Set that is to be associated with the tag. An alarm point is required to have a digital set and the digital set must be created before the creation of the alarm point. See Section 8.3, *Alarm State Sets*, for more information about Alarm Digital States.

8.2.6 ReferenceTag

The tagname defined in the **ReferenceTag** attribute is used in the **Test1**, **Test2, Test3**, and **Test4** attributes as a reference point to trigger alarms. In the case where a reference is used, the argument in the test is the keyword *ref*. In this case, the **ReferenceTag** attribute must be the tagname of the desired reference point.

8.2.7 AutoAck

The two options that can be set for the **AutoAck** attribute are **yes** or **no**. The default is **yes**. If the value in **AutoAck** attribute is **yes** then an alarm is automatically acknowledged. For a three-acknowledgment status configuration in the Alarm State Set, the new and unacknowledged missed statuses shown in Table 8–12 would never be assigned to an alarm point value.

8.2.8 DeadBand

The **DeadBand** attribute modifies the **Test1**, **Test2**, **Test3**, and **Test4** attributes of **GT** and **LT**. The **DeadBand** is a threshold, within the alarm limit, that the rate point must pass after an alarm is triggered before the point is considered not to be in alarm. The default **DeadBand** is **0**.

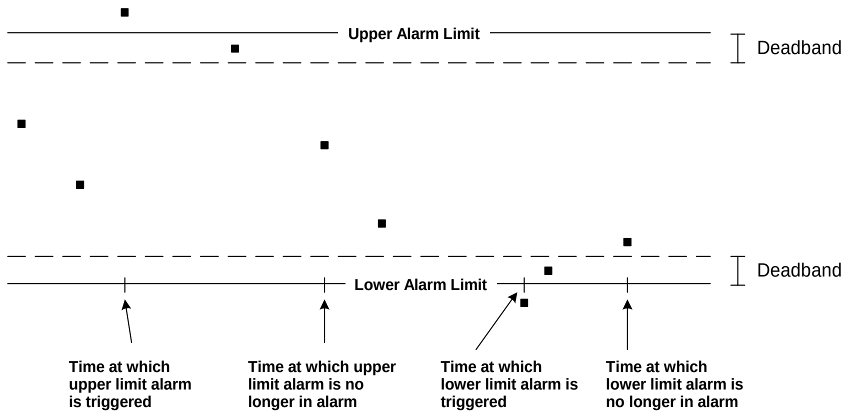

Upper Alarm Limit
Deadband
Deadband
Lower Alarm Limit
Time at which upper limit alarm is triggered
Time at which upper limit alarm is no longer in alarm
Time at which lower limit alarm is triggered
Time at which lower limit alarm is no longer in alarm

Figure 8–3. Deadbands for Upper and Lower Alarm Limits

8.2.9 Options

The option attribute is used to modify the period associated with the **RateGT** and **RateLT** operators in the **Test1**, **Test2**, **Test3**, and **Test4** attributes.

```
Syntax:RTime = timestamp
```

The following examples set the period to 1 hour, 5 minutes, and 30 seconds respectively.

```
RTime = +1h
Rtime = +5m
Rtime = +30s
```

8.2.10 ControlTag

The tagname defined in the **ControlTag** attribute is used in the combiner logic to disable alarms. The tagname specified by the **ControlTag** attribute must refer to a numeric (any float or integer) or digital point type. Any other point type will result in an **Error** state for the alarm point. In the case where a control tag is used, the alarm can be taken out of service by setting the value of the control tag to zero for numeric tags or the **0th** state for digital tags.

Below is an example of using the **ControlTag** to disable an alarm when net generation drops below a predetermined value. This could easily be applied to all alarms on a unit. Assume that *U2:NetGen.PV* is the net generation on our example unit.

Configure a PE point, a digital point, with 2 states: *Not Running* and *Running*. Be sure that *Not Running* is first digital state in the set. Set the point to be naturally scheduled, based on our net generation value. The **ExDesc** of the PE will look similar to:

```
event=U2:NetGen.PV, if('U2:NetGen.PV' > 5) THEN "Running" ELSE "Not Running"
```

If there are only 2 states in this digital set, and *Not Running* is the first one, the following **ExDesc** will do the same thing.

```
event=U2:NetGen.PV, 'U2:NetGen.PV' > 5
```

Finally, enter the tagname of this PE as the **ControlTag** of any alarms that should be disabled when net generation falls below **5**.

8.2.11 ControlAlg

Reserved attribute for future implementation.

8.3 Alarm State Sets

An Alarm State Set is a Digital State Set constructed such that **condition**, **acknowledgement**, and **priority** codes are all encoded in the digital states.

8.3.1 Condition

The condition of an alarm describes the manner in which an alarm is manifested. Table 8–11 shows some sample descriptions of the types of alarm conditions that may be implemented.

Table 8–11. Sample Alarm Conditions

Condition	Example Description
Lolo	The value of the point is way below normal
Low	The value of the point is below normal
High	The value of the point is above normal
Hihi	The value of the point is way above normal
Rate	The rate of change of the point is abnormal
Step	The change in value of the point is abnormal
Change	The value of the point has changed from the previous

8.3.2 Acknowledgement Status

The acknowledgement shows the status or the amount of attention that has been paid to the alarm. The acknowledgement status is a modifier to the condition of the alarm. This release of PI Alarm allows for 1 or 3 states of acknowledgement statuses. The three-state acknowledgement status set consists of the *New Alarm*, the *Acknowledged Alarm* and *Unacknowledged Alarm* that is missed. The one-state acknowledgement is defined by having all alarms acknowledged. Table 8–12 shows the three-state acknowledgement status and respective descriptions.

Table 8–12. Three-state Acknowledgement Status

Acknowledgment Status	Description
New	An alarm has been triggered and is unacknowledged.
Acknowledged	The triggered alarm is acknowledged.
Unacknowledged Missed	The unacknowledged alarm was missed because the source point is no longer in alarm.

8.3.3 Priority

The priority describes the level of importance given to the triggered alarm or the severity associated with the triggered alarm. Table 8–13 is an Alarm Digital State Set with a single priority and three acknowledgement statuses.

Table 8–13. Single Priority Alarm State Set

New Alarm	Acknowledged Alarm	Unacknowledged Alarm Missed
Lolo <<	Lolo	Lolo _x
Low <<	Low	Low _x
High <<	High	High _x
Hihi <<	Hihi	Hihi _x
Rate <<	Rate	Rate _x
Step <<	Step	Step _x
Change <<	Change	Change _x
Dig1 <<	Dig1	Dig1 _x
Dig2 <<	Dig2	Dig2 _x

In this scheme, the acknowledgement statuses are represented by symbols. New alarms are denoted by the symbol “<<” at the end of the alarm conditions, while acknowledged alarms are just denoted as the alarm conditions itself. Alarms that are missed because the source point has gone out of an alarm condition are denoted by an underscore followed by an “x” (“_x”) after the alarm condition.

For the case where more severity is placed on certain types of alarms as opposed to others, the priority level can be expanded to multiple levels. Table 8–14 shows an example of a three-priority system with their respective meanings.

Table 8–14. Example Three-priority-level System

Priority Level	Example Description	Sample Symbol
1	Alert	__
2	Important	_*
3	Most Urgent	**

The three priorities are classified from the most severe (priority level 3) to the least severe (priority level 1) as **Most Urgent**, **Important**, and **Alert** and their respective symbols which is denoted in front of the alarm condition are "**", "_*", and "__" respectively. Table 8–15 shows the same Alarm Digital State Set with the additional priorities.

Table 8–15. Alarm Digital State Set with Three Priorities

Priority	New alarm	Acknowledged Alarm	Unacknowledged Alarm Missed
Most Urgent	** Lolo <<	** Lolo	** Lolo _x
	** Low <<	** Low	** Low _x
	** High <<	** High	** High _x
	** Hihi <<	** Hihi	** Hihi _x
	** Rate <<	** Rate	** Rate _x
	** Step <<	** Step	** Step _x
	** Change <<	** Change	** Change _x
	** Dig1 <<	** Dig1	** Dig1 _x
	** Dig2 <<	** Dig2	** Dig2 _x
Important	_* Lolo <<	_* Lolo	_* Lolo _x
	_* Low <<	_* Low	_* Low _x
	_* High <<	_* High	_* High _x
	_* Hihi <<	_* Hihi	_* Hihi _x
	_* Rate <<	_* Rate	_* Rate _x
	_* Step <<	_* Step	_* Step _x
	_* Change <<	_* Change	_* Change _x
	_* Dig1 <<	_* Dig1	_* Dig1 _x
	_* Dig2 <<	_* Dig2	_* Dig2 _x
Alert	__ Lolo <<	__ Lolo	__ Lolo _x
	__ Low <<	__ Low	__ Low _x
	__ High <<	__ High	__ High _x

Priority	New alarm	Acknowledged Alarm	Unacknowledged Alarm Missed
	__ Hihi <<	__ Hihi	__ Hihi _x
	__ Rate <<	__ Rate	__ Rate _x
	__ Step <<	__ Step	__ Step _x
	__ Change <<	__ Change	__ Change _x
	__ Dig1 <<	__ Dig1	__ Dig1 _x
	__ Dig2 <<	__ Dig2	__ Dig2 _x

The only default Alarm State Set that is installed is the **Pialarm33 State Set**. Four sample Alarm State Sets are also included in a **piconfig** script (two for use with numeric source tags and two for digital). These sample Alarm State Sets are provided in Section 8.3, *Alarm State Sets*. For each type, one set is simple, only showing alarm **condition**, and the other complete, showing **priority**, **condition**, and **acknowledge** status. The sample *Alarm State Sets for numeric source points* are displayed in Table 8–13 and Table 8–14. A second set of *Alarm State Sets for digital or string source points* are provided in Table 8–16 and Table 8–17.

Table 8–16. Example Digital Alarm State Set with One Priority

New alarm	Acknowledged Alarm	Unacknowledged Alarm Missed
Move <<	Move	Move _x
Fail <<	Fail	Fail _x
Off <<	Off	Off _x
Over <<	Over	Over _x
Under <<	Under <<	Under <<
Change <<	Change	Change _x
D7 <<	D7	D7 _x
D8 <<	D8	D8 _x

Table 8–17. Example Digital Alarm Set with Three Priorities

Priority	New alarm	Acknowledged Alarm	Unacknowledged Alarm Missed
	** Move <<	** Move	** Move _x

Priority	**New alarm**	**Acknowledged Alarm**	**Unacknowledged Alarm Missed**
	** Fail <<	** Fail	** Fail _x
	** Off <<	** Off	** Off _x
	** Over <<	** Over	** Over _x
	** Under <<	** Under	** Under _x
	** Change <<	** Change	** Change _x
	** D7 <<	** D7	** D7 _x
	** D8 <<	** D8	** D8 _x
Important	_* Move <<	_* Move	_* Move _x
	_* Fail <<	_* Fail	_* Fail _x
	_* OFF <<	_* OFF	_* OFF _x
	_* Over <<	_* Over	_* Over _x
	_* Under <<	_* Under	_* Under _x
	_* Change <<	_* Change	_* Change _x
	_* D7 <<	_* D7	_* D7 _x
	_* D8 <<	_* D8	_* D8 _x
Alert	__ Move <<	__ Move	__ Move _x
	__ Fail <<	__ Fail	__ Fail _x
	__ OFF <<	__ OFF	__ OFF _x
	__ Over <<	__ Over	__ Over _x
	__ Under <<	__ Under	__ Under _x
	__ Change <<	__ Change	__ Change _x
	__ D7 <<	__ D7	__ D7 _x
	__ D8 <<	__ D8	__ D8 _x

Each site using PI Alarms needs to examine its needs for alarm display sets and establish local standards. While state sets can be added or changed, some changes will require significant re-configuration of client programs. This can be minimized by planning.

As with all Digital State Sets, each of the digital state strings in the tables corresponds to a digital state code. The manner in which the Alarm State Sets mentioned above are changed into digital states is explained in Section 8.11, *Alarm State Set Encoding and Decoding*.

8.4 Alarm Groups

An Alarm Group is a collection of alarm points that are grouped together using a common group name. Furthermore, Alarm Groups may also contain other Alarm Groups, to create an alarm hierarchy. Each alarm point is assigned to a single Alarm Group. The Alarm Group organization should match the operational structure of the plant. This hierarchy forms the basis for alarm system control and reporting functions.

Each Alarm Group is represented in the PI system by one or more points. A single point is required to provide the name for the group referred to by alarm points within it. The Alarm Group point can receive values that represent statistics about the alarms in the group. Additional points allow the alarm subsystem to report group statistics (number of alarms, number of unacknowledged alarms, etc.). These additional statistical points are optional.

8.4.1 Alarm Group Point Configuration

An Alarm Group is represented by one or more points in the PI System. The default point source for Alarm Group points is **G**. The Alarm Group uses the point class base with a numeric data type. The parameters that configure an Alarm Group are contained in the **ExDesc** attribute.

Attribute	Parameters
ExDesc	GroupName PointFunc Arg

The **GroupName** parameter defines the name of the group. The **PointFunc** parameter contains the options to be set for the group. Statistics of the Alarm Group are described in the **PointFunc** table (Table 8–18). The **Arg** parameter is a modifier to the **PointFunc** parameter. Available statistics include the number of alarms at each priority and the highest priority of the currently active alarms in the group. Groups can be arranged on a hierarchy.

Table 8–18. PointFunc Options

PointFunc	Description
GroupID	Setting **PointFunc** to **GroupID** defines the Alarm Group. The optional **Arg** parameter is the name of the group's parent group in the hierarchy (if it exists)
UnAck	Displays total number of unacknowledged alarms within a group or priority. The **Arg** parameter defines the priority for the unacknowledged alarms.
InAlarm	Displays total number of points in alarm within a group or priority. The **Arg** parameter defines the priority of the alarms.

In that case, the statistics at the parent level include the points in the groups below. Groups can be arranged to any depth. There is nothing preventing both points and groups from being assigned to the same group. Table 8–19 shows an example Alarm Group hierarchy with two subgroups.

Table 8–19. Example Alarm Group

TagName	ExDesc	Description
AlarmTop	AlarmTop GroupID	Name of Alarm Group for the top of alarm hierarchy and also total number of alarm points within the hierarchy in an Alarm State.
Tot_Unack	AlarmTop UnAck	Total number of unacknowledged alarms in AlarmTop hierarchy
AlarmGrp1	AlarmGrp1 GroupID AlarmTop	Name of Alarm Group that belongs to the AlarmTop hierarchy and also the total number of alarm points in this group that is in an Alarm State.
AlarmGrp1_Unack_1	AlarmGrp1 UnAck 1	Number of unacknowledged alarms in AlarmGrp1 with priority 1
AlarmGrp1_InAlarm_1	AlarmGrp1 InAlarm 1	Number of alarm points in an Alarm State in AlarmGrp1 with priority 1
AlarmGrp2	AlarmGrp2 GroupID AlarmTop	Name of Alarm Group that belongs to the AlarmTop hierarchy and also the total number of alarm points in this group that is in an Alarm State.
AlarmGrp2_InAlarm_0	AlarmGrp2 InAlarm 0	Number of alarm points in an Alarm State in AlarmGrp2 with priority 0
AlarmGrp2_Unack	AlarmGrp2 UnAck	Total number of unacknowledged alarms in AlarmGrp2

This example Alarm Group contains the point *AlarmTop* that defines the top of the alarm hierarchy. *AlarmTop* and *Tot_Unack* give the statistics of the total number of alarms points that are in alarm and the total number of unacknowledged alarms in the alarm hierarchy that are in the alarm hierarchy. *AlarmGrp1* and *AlarmGrp2* are the two Alarm Groups within the hierarchy.

AlarmGrp1_Unack_1 and **AlarmGrp1_InAlarm_1** are the statistics of the number of unacknowledged alarms in Alarm Group *AlarmGrp1* with priority **1** and the number of alarm points in *AlarmGrp1* that are in alarm with priority **1**, respectively.

AlarmGrp2_InAlarm_0 and **AlarmGrp2_Unack** are the number of alarm points in an Alarm State with priority **0** in *AlarmGrp2*, and the statistics of the total number of unacknowledged alarms in *AlarmGrp2*, respectively. Using the modifier Arg of **'UnAck'** with the priority **0** will always return **0** because conditions are always considered acknowledged.

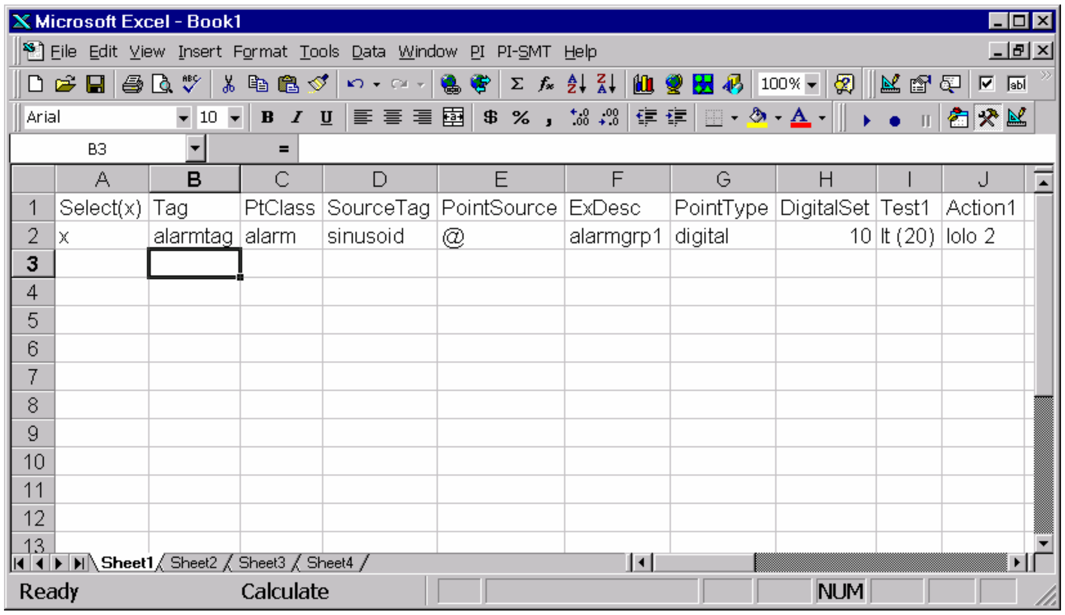
8.5 Build Alarm Points

Alarm points are created in the same manner as other points except that the Alarm must be of alarm point class, which includes Base attributes. The point type must be digital and the **PointSource** must be '@'.

To override the **PointSource** default values, see *Override Default PointSource Values for Alarms*, page 330.

8.5.1 PI TagConfigurator

You can easily create alarm points with the **PI TagConfigurator** tool. See the Rockwell Automation Technical Support Web site to download this tool.



	A	B	C	D	E	F	G	H	I	J
1	Select(x)	Tag	PtClass	SourceTag	PointSource	ExDesc	PointType	DigitalSet	Test1	Action1
2	x	alarmtag	alarm	sinusoid	@	alarmgrp1	digital	10	lt (20)	lolo 2

Figure 8–4. PI TagConfigurator Creation of Alarm Point

8.5.2 Piconfig

Alternatively, you can use **piconfig** to create an alarm point:

```
@table pipoint
@ptclass alarm
@mode create, t
@istru tag, pointsource, sourcetag, exdesc, pointtype, digitalset, test1,
action1
alarmtag1, @, sinusoid, alarmgrp1, digital, pialarm33, lt (20), lolo 2
@ends
```

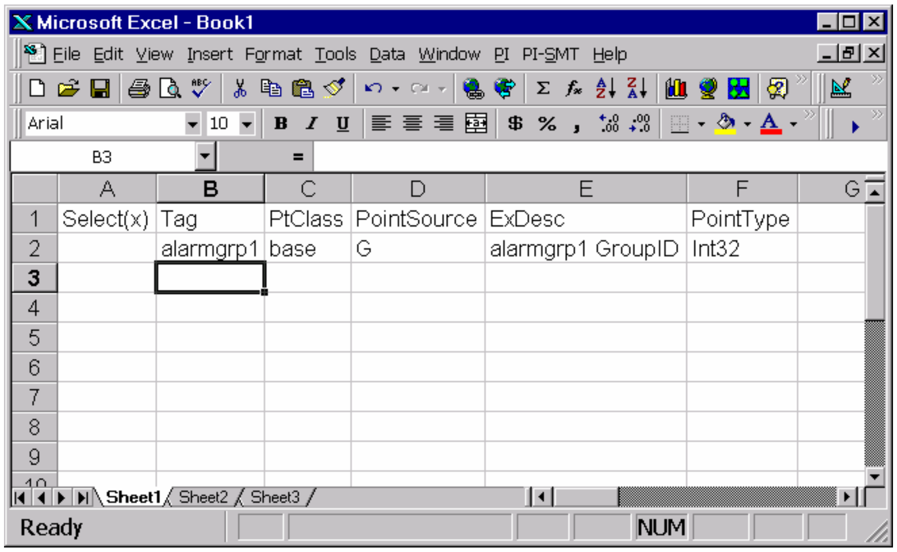
8.6 Build Alarm Group Points

Alarm Group points are created in the same manner as other points. The point type must be numeric and the **PointSource** must be **G**.

To override the **PointSource** default values, see *Override Default PointSource Values for Alarms*, page 330.

8.6.1 PI TagConfigurator

Alarm Group points can be created using the PI Tag Configurator utility, which you can obtain from the Rockwell Automation support Web site.


Microsoft Excel - Book1
File Edit View Insert Format Tools Data Window PI PI-SMT Help
Arial
10
B3
=
Sheet1
Sheet2
Sheet3
Ready
NUM

	A	B	C	D	E	F	G
1	Select(x)	Tag	PtClass	PointSource	ExDesc	PointType	
2		alarmgrp1	base	G	alarmgrp1 GroupID	Int32	
3							
4							
5							
6							
7							
8							
9							

Figure 8–5. PI Tag Configurator

8.6.2 Piconfig

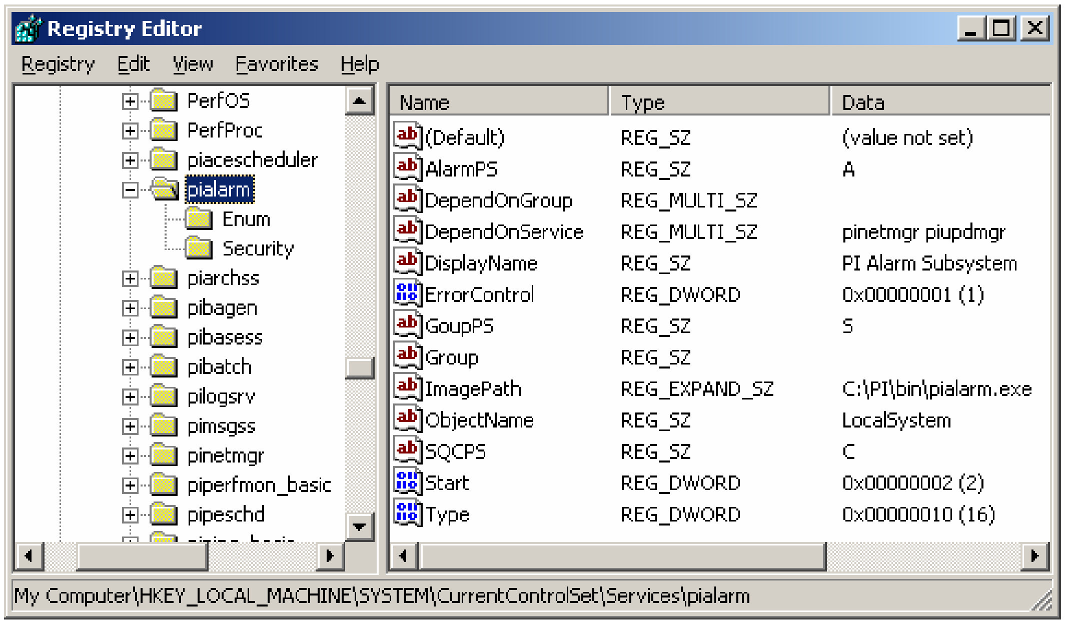
Alternatively, you can use the **piconfig** utility to create the same Alarm Group point.

```
@table pipoint
@ptclass base
@mode create, t
@istru tag, pointsource, exdesc, pointtype
alarmgrp1, G, “alarmgrp1 GroupID”, Int32
@ends
```

8.7 Override Default PointSource Values for Alarms

You can override the default **PointSource** values by setting particular values in the Windows Registry. Under the Registry key, **SYSTEM/CurrentControlSet/Services/pialarm,** the string values *AlarmPS*, *GroupPS*, and *SqcPS* will be present if used.

For example, in Figure 8–6, the RegEdit screen has been reset with the **PointSource** for **Alarm points** set to **A**, the **PointSource** for **group points** to **S**, and the **PointSource** for **sqc alarmpoints** to **C**. Changes take effect only when the **pialarm** service is restarted.



PerfOS
PerfProc
piacescheduler
pialarm
Enum
Security
piarchss
pibagen
pibasess
pibatch
pilogsrv
pimsgss
pinetmgr
piperfmon_basic
pipeschd

Name	Type	Data
(Default)	REG_SZ	(value not set)
AlarmPS	REG_SZ	A
DependOnGroup	REG_MULTI_SZ	
DependOnService	REG_MULTI_SZ	pinetmgr piupdmgr
DisplayName	REG_SZ	PI Alarm Subsystem
ErrorControl	REG_DWORD	0x00000001 (1)
GoupPS	REG_SZ	S
Group	REG_SZ	
ImagePath	REG_EXPAND_SZ	C:\PI\bin\pialarm.exe
ObjectName	REG_SZ	LocalSystem
SQCPS	REG_SZ	C
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\pialarm

Figure 8–6. Editing Default PointSource Values for Alarms

For UNIX systems, the command line that starts the Alarm Subsystem may contain optional arguments to override the default point sources. For the same settings as in the Windows example above, include the following options in the command line: `-ps=A`, `-gps=S`, and `-qps=C`.

8.8 Build Alarm Digital State Sets

Alarm Digital State Sets are similar to all digital state sets in that each digital state string must correspond to a digital state code. The Alarm State Set is special because there is a specific structure that is required for this type of digital state set. The following **piconfig** script is an example that shows how to build a simple Alarm Digital State Set with three acknowledgement statuses, three priorities, and two conditions - *low* and *high*.

```
@table PIDS
@mode create,t
@istyp delim,
@istru set,
@istru oldcode ,state
@istru ...,
pialm33,
1, ----
2, __ Low ^^
3, _* Low ^^
4, ** Low ^^
5, __ Low <<
6, _* Low <<
```

```
7, ** Low <<
8, __ Low _x
9, _* Low _x
10, ** Low _x
11, __ High ^^
12, _* High ^^
13, ** High ^^
14, __ High <<
15, _* High <<
16, ** High <<
17, __ High _x
18, _* High _x
19, ** High _x
20, LOW
21, HIGH
22, 3 3
@ends,
```

In the above script, the Alarm State Set is named *pialm33*. There are 22 digital states. The first digital state is the state of *no alarm*. In this script, the *no alarm status* is represented by the symbol “----”.

```
1, ----
```

If the desired representation for *no alarm* is “No Alarm,” this line in the script would be:

```
1, No Alarm
```

The last digital state (22nd) corresponds to the number of acknowledgment statuses and the number of priorities that exist for the Alarm State Set. In this example, it is three for both. For only one priority, the line would read as follows:

```
22, 3 1
```

The Alarm State Set is created in a manner similar to having three nested loops, where the outer loop is the condition, the middle loop is the acknowledgement status, and the innermost loop is the priority. So for each condition, the innermost loop sets the priorities from least severe to most severe, which are represented here by the symbols “**__**”, “**_***”, “******”.

```
2, __ Low ^^
3, _* Low ^^
4, ** Low ^^
```

The middle loop then sets the acknowledgement statuses in the order of acknowledged alarm (“ ^^“), new alarm (“<<”) and unacknowledged alarm missed (“_x”).

```
2, __ Low ^^
3, _* Low ^^
4, ** Low ^^
5, __ Low <<
6, _* Low <<
7, ** Low <<
```

```
8, __ Low _x
9, _* Low _x
10, ** Low _x
```

The outer loop finally loops around the conditions. Hence each condition has 3 acknowledgement statuses and 3 priorities, and thus 9 digital states each. Since the first digital state code is the *no alarm condition*, digital state codes 2 though 19 are used for the Alarm States. At the end of the Alarm States are the condition names themselves.

```
20, LOW
21, HIGH
```

As mentioned earlier, the last digital state code is used to tell the alarm program how many acknowledgement statuses and priorities are in the Alarm State Set.

```
22, 3 3
```

Further details on encoding and decoding Alarm Digital States is provided in *Alarm State Set Encoding and Decoding*, page 335.

8.9 Program Operation

8.9.1 Startup

The Alarm Subsystem program is started with the rest of the system. The location of the program, **pialarm.exe** for Windows and **pialarm** for UNIX, is in the *pi\bin* directory. All rate calculations are reset when the PI System or the Alarm Subsystem is restarted.

8.9.2 Alarm Notification

Alarm Points are standard digital points. Client programs can request to be alerted when point values change through the standard Update Manager mechanisms. FactoryTalk Historian ProcessBook can use alarm summary tags as the basis for the operator alert process without signing up for every alarm point in the system. Future versions of the alarm server will allow a client to receive alarm updates on an Alarm Group basis.

Viewing Alarms in FactoryTalk Historian ProcessBook

The following figure shows an example of an alarm display in FactoryTalk Historian ProcessBook.

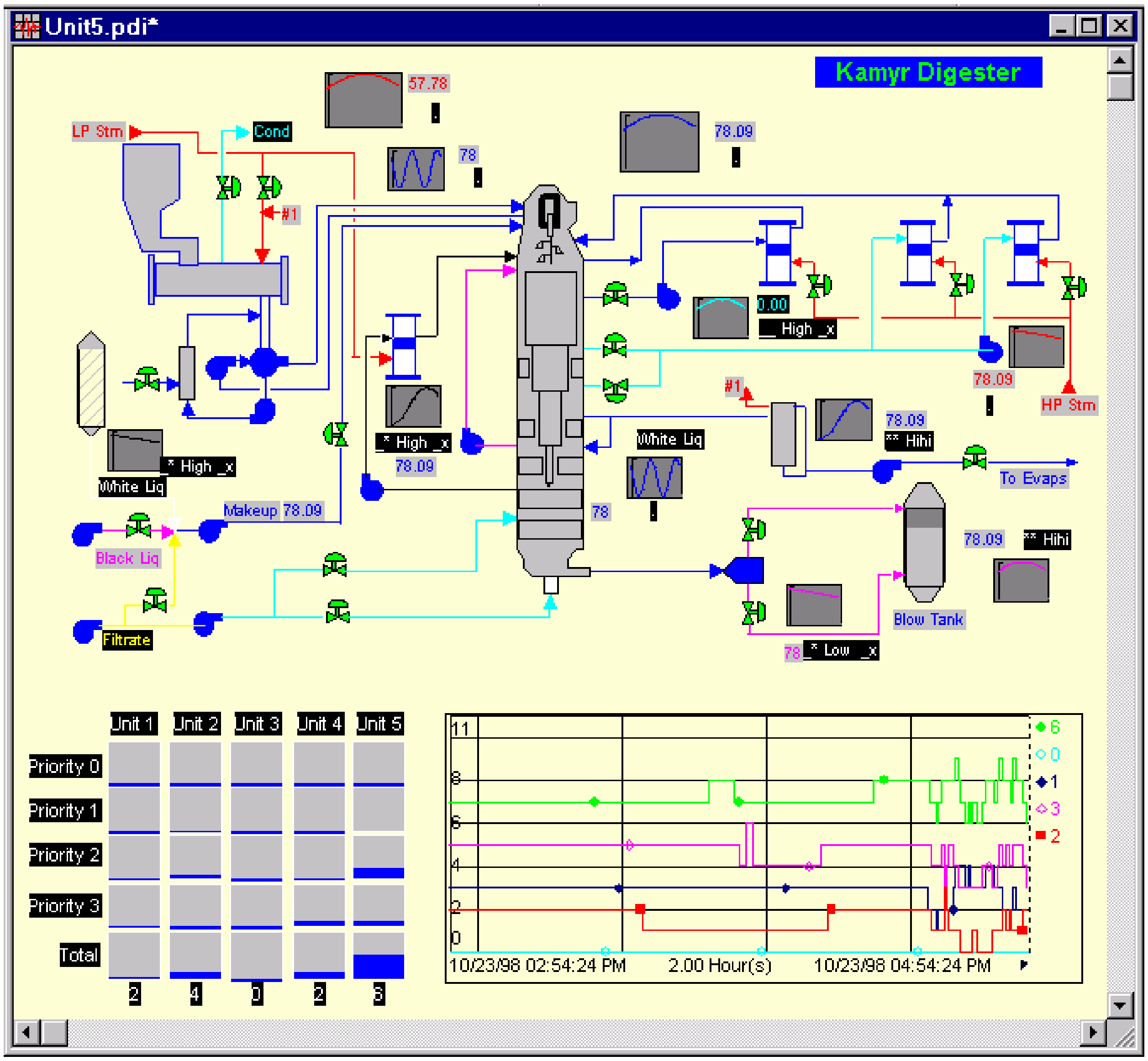


Figure 8–7. Example FactoryTalk Historian ProcessBook Display of Alarms

The example above is edited from the Kamyr display, from the *PIDemo* example included in PIProcessBook. It shows the alarm point values next to the values of the source tag as well as a trend display of the statistics of the alarms for the unit (Unit5). The chart on the bottom left corner displays the statistics of the alarms for the four other units as well as the current unit.

8.9.3 Error Messages

All significant events in the life of an alarm and Alarm Group point are noted in the system Message Log, including point additions, edits and error conditions. For UNIX, an additional log file is called *pilarm.log.* To monitor logged events, use the **pigetmsg** utility. See the ***PI Server System Management Guide*** for information about using the **pigetmsg** utility.

8.9.4 Demonstration Points

Demonstration points for the PI Alarm Subsystem are available in the *pi\adm* directory in the *Alarmpts.dif* file but are not automatically created when the system is installed. These alarm demonstration points can be created by redirecting the file to the **piconfig** utility. See Chapter 11, *The Piconfig Utility*, in the ***PI Server System Management Guide*** for information about using the **piconfig** utility.

```
$ piconfig < alarmpts.dif
```

8.10 PI for OpenVMS Upgrade Considerations

The PI Server Alarm subsystem is capable of having all the alarms of the PI for OpenVMS version and more.

The following are known differences:

- The method of alarm messaging is not supported on the server side. The Alarm Client program will allow for the viewing of alarms and Alarm Groups as well as messaging. This feature will be available in the future.
- A log of the alarms is not available. The history of the alarm is contained in the alarm points themselves.
- Acknowledgement of the alarm is accomplished by selecting the auto-acknowledgement feature or writing an *acknowledged digital state* to the PI system through the PI-API.

8.10.1 New Alarm Subsystem Features in PI Server 3.4

The Alarm Subsystem includes new features that are not included in PI for OpenVMS.

- The Alarm Subsystem allows you to set Alarm priorities.
- Alarm Groups can incorporate other Alarm Groups, to form hierarchies.
- The Alarm Subsystem supports testing of string tags.

8.11 Alarm State Set Encoding and Decoding

Due to the manner in which the Alarm State Set is structured, conversion from the digital state code or offset to the condition, acknowledgement status, and priority is not as straightforward as it may seem.

There is a distinction between the manner in which Digital State Sets are created and the digital state code (offset) that is returned from a client program. For example, the following is part of the **piconfig** script to create an Alarm Digital State Set as explained in Section 8.8, *Build Alarm Digital State Sets*.

```
@table PIDS,
@mode create,t
@istyp delim,
@istru set,
@istru oldcode ,state
@istru ...,
pialm33,
1, ----
2, __ Low ^^
3, _* Low ^^
4, ** Low ^^
5, __ Low <<
etc…
```

In the script, the digital states are numbered starting at **1**. On the contrary, the offset, which is returned from a client program returns the digital state offset in which the number begins at **0**. Therefore, for encoding and decoding, the digital state code is the digital state offset. Hence the offset for the digital state shown for `** Low ^^` in the script is **3** and not **4**.

In order for client applications to convert from digital state codes to the condition, acknowledgement status, and priority, and vice versa, the following algorithms should be applied. See section 8.8, *Build Alarm Digital State Sets* for more information.

8.11.1 Conversion to Digital State code

The algorithm for conversion points first needs to change the condition, **C**, acknowledgement status, **A**, and priority, **P**, to numeric values. In the conversion the value for the condition is given by where the condition is placed in the Alarm State Set. For example, in the Alarm State Set given in Table 8–15, the conditions are listed as follows.

```
Lolo
Low
High
Hihi
Rate
Step
Change
Dig1
Dig2
```

Therefore, the position for the condition **Hihi** within the list of conditions is **4**, and the number of conditions, **MaxC**, is **9**. **Nack** is the number of acknowledgement statuses. This number can only be **1** or **3**. The numeric value for the acknowledgement statuses, **A**, for the case when **Nack** is **1** will be **1**. The numeric values for the situation when **Nack** is **3**, is given as follows.

```
A = 0 (Acknowledged)
A = 1 (New Alarm)
A = 2 (Unacknowledged Missed)
```

The priority, **P**, is level of severity associated with the alarm. This number increases as the severity of the alarm increases and goes from **0** to the maximum priority, **MxP**.

For example, an Alarm State Set with **MxP** equal to **3** is as follows.

```
P = 0 (Condition Only)
P = 1 (Alert)
P = 2 (Important)
P = 3 (Most Urgent)
```

P equal to **0** results in the alarm condition. The number of states per condition **Pa1** is given by multiplying the number of acknowledgeable statuses by the **max** numeric priority level:

```
Pa1 = Nack * MxP
```

For example, if the two conditions in the Alarm State Set are high and low, then **MaxC** is equal to **2**. The number of states in the Alarm State Set is **Nsts**.

```
Nsts = Pa1 * MaxC + MaxC
```

The digital state code, **code**, is then given by the following equation.

```
code = Pa1 * ( C - 1 ) + Mxp * A + P
```

8.11.2 Conversion from Digital State Code

Given the numeric values for the condition, **C**, acknowledgement status, **A**, and Priority, **P**, the digital state code, **code**, is obtained using the following equations points.

```
C = int( ( code-1)/Pa1 ) +1
A = int( ( code-Pa1*(C-1)-1)/MxP )
P = code-Pa1*(C-1) - MxP * A
```

In the prior equations, the number of states per condition is **Pa1** and the maximum number of priorities is **MxP**. These are defined in Section 8.8, *Build Alarm Digital State Sets*.

8.11.3 Sample Alarm Digital State Sets

The following tables list the four Alarm State Sets and their respective digital state numbers, which are used in a **piconfig** session points. The only Alarm Digital State Set that is installed (by default) is **pialarm33**.

These Alarm State Sets can be created in a **piconfig** session using the file *almstate.dif*.

```
$ piconfig < almstate.dif
```

One Priority Alarm Set

Table 8–20. One Priority Alarm Set

Offset	AlarmSet
0	.
1	Lolo
2	Low
3	High
4	Hihi
5	Rate

Offset	AlarmSet
6	Change
7	Dig1
8	Dig2
9	Lolo <<
10	Low <<
11	High <<
12	Hihi <<
13	Rate <<
14	Change <<
15	Dig1 <<
16	Dig2 <<
17	Lolo _x
18	Low _x
19	High _x
20	Hihi _x
21	Rate _x
22	Change _x
23	Dig1 _x
24	Dig2 _x
25	LOLO
26	LOW
27	HIGH
28	HIHI
29	RATE
30	CHANGE
31	DIG1
32	DIG2
33	3 1

Three Priority Alarm Set

Table 8–21. Three Priority Alarm Set

Offset	AlarmSet
0	.

Offset	**AlarmSet**
1	__ Lolo
2	_* Lolo
3	** Lolo
4	__ Lolo <<
5	_* Lolo <<
6	** Lolo <<
7	__ Lolo _x
8	_* Lolo _x
9	** Lolo _x
10	__ Low
11	_* Low
12	** Low
13	__ Low <<
14	_* Low <<
15	** Low <<
16	__ Low _x
17	_* Low _x
18	** Low _x
19	__ High
20	_* High
21	** High
22	__ High <<
23	_* High <<
24	** High <<
25	__ High _x
26	_* High _x
27	** High _x
28	__ Hihi
29	_* Hihi
30	** Hihi
31	__ Hihi <<
32	_* Hihi <<
33	** Hihi <<

Offset	**AlarmSet**
34	__ Hihi _x
35	_* Hihi _x
36	** Hihi _x
37	__ Rate
38	_* Rate
39	** Rate
40	__ Rate <<
41	_* Rate <<
42	** Rate <<
43	__ Rate _x
44	_* Rate _x
45	** Rate _x
46	__ Step
47	_* Step
48	** Step
49	__ Step <<
50	_* Step <<
51	** Step <<
52	__ Step _x
53	_* Step _x
54	** Step _x
55	__ Change
56	_* Change
57	** Change
58	__ Change <<
59	_* Change <<
60	** Change <<
61	__ Change _x
62	_* Change _x
63	** Change _x
64	__ Dig1
65	_* Dig1
66	** Dig1

Offset	**AlarmSet**
67	__ Dig1 <<
68	_* Dig1 <<
69	** Dig1 <<
70	__ Dig1 _x
71	_* Dig1 _x
72	** Dig1 _x
73	__ Dig2
74	_* Dig2
75	** Dig2
76	__ Dig2 <<
77	_* Dig2 <<
78	** Dig2 <<
79	__ Dig2 _x
80	_* Dig2 _x
81	** Dig2 _x
82	LOLO
83	LOW
84	HIHI
85	HIGH
86	RATE
87	STEP
88	CHANGE
89	DIG1
90	DIG2
91	3 36

8.11.4 Digital Base Set

Single Priority Alarm Set

Table 8–22. Single Priority Alarm Set (Digital Base Set)

Offset	**AlarmSet**
0	.
1	Move
2	Fail

Offset	AlarmSet
3	OFF
4	Over
5	Under
6	Change
7	Dig6
8	Dig7
9	Move <<
10	Fail <<
11	OFF <<
12	Over <<
13	Under <<
14	Change <<
15	Dig7 <<
16	Dig8 <<
17	Move _x
18	Fail _x
19	OFF _x
20	Over _x
21	Under _x
22	Change _x
23	Dig7 _x
24	Dig8 _x
25	MOVE
26	FAIL
27	OFF
28	OVER
29	UNDER
30	CHANGE
31	DIG7
32	DIG8
33	3 1

Three Priority Alarm Set

Table 8–23. Three Priority Alarm Set (Digital Base Set)

Offset	AlarmSet
0	.
1	__ Move
2	_* Move
3	** Move
4	__ Move <<
5	_* Move <<
6	** Move <<
7	__ Move _x
8	_* Move _x
9	** Move _x
10	__ Fail
11	_* Fail
12	** Fail
13	__ Fail <<
14	_* Fail <<
15	** Fail <<
16	__ Fail _x
17	_* Fail _x
18	** Fail _x
19	__ OFF
20	_* OFF
21	** OFF
22	__ OFF <<
23	_* OFF <<
24	** OFF <<
25	__ OFF _x
26	_* OFF _x
27	** OFF _x
28	__ Over
29	_* Over
30	** Over

Offset	AlarmSet
31	__ Over <<
32	_* Over <<
33	** Over <<
34	__ Over _x
35	_* Over _x
36	** Over _x
37	__ Under
38	_* Under
39	** Under
40	__ Under <<
41	_* Under <<
42	** Under <<
43	__ Under _x
44	_* Under _x
45	** Under _x
46	__ Change
47	_* Change
48	** Change
49	__ Change <<
50	_* Change <<
51	** Change <<
52	__ Change _x
53	_* Change _x
54	** Change _x
55	__ Dig7
56	_* Dig7
57	** Dig7
58	__ Dig7 <<
59	_* Dig7 <<
60	** Dig7 <<
61	__ Dig7 _x
62	_* Dig7 _x
63	** Dig7 _x

Offset	**AlarmSet**
64	__ Dig8
65	_* Dig8
66	** Dig8
67	__ Dig8 <<
68	_* Dig8 <<
69	** Dig8 <<
70	__ Dig8 _x
71	_* Dig8 _x
72	** Dig8 _x
73	MOVE
74	FAIL
75	OFF
76	OVER
77	UNDER
78	CHANGE
79	DIG7
80	DIG8
81	3 3

Chapter 9. PI REAL-TIME SQC

Statistical Quality Control (SQC) is the use of numerical methods to monitor the characteristics of a process, making sure they remain within pre-determined boundaries.

The **PI Real-Time SQC** (**PI SQC**) component allows you to apply Western Electric Pattern Tests to all process or laboratory data collected by the PI System. PI SQC continually reviews any SQC tests in the PI System. It stores test results, and a record of SQC control limits back into your PI System. The results are available for viewing and analysis via FactoryTalk Historian ProcessBook and the PI SQC Add-In.

PI SQC is a part of PI Alarm, which provides continual evaluation of SQC pattern tests and the management of alarms generated from them. When an unacceptable deviation from test norm occurs, PI SQC notifies the PI SQC Alarm Manager.

The chapter includes an overview of SQC fundamentals, an understanding of how PI SQC works, an explanation of how to configure the PI points on which PI SQC functionality depends, and instructions on how to install PI SQC.

The following topics are included:

The following books, on the subject of Statistical Quality Control (SQC), were used in developing this chapter, and may be helpful in developing your implementation of PI SQC.

- Wheeler, Donald J., *Advanced Topics in Statistical Process Control: The Power of Shewart's Charts*, 1995, SPC Press, Knoxville
- *Statistical Quality Control Handbook*, 2nd Edition, 1958, Western Electric Company

9.1 Introduction to Statistical Quality Control

Whenever a series of observations or measurements of a process parameter are examined the measurements will not, in general, be identical to each other. Statistical Quality Control is beased on the core premise that all processes fluctuate. The fluctuations may be natural or unnatural. Natural fluctuations are generally small, while unnatural fluctuations are larger and introduced by external (hopefully, definable) causes. SQC provides a set of simple tools to identify instances of unnatural fluctuation so that causes can be assigned and corrected.

In simple terms:

- *Everything Varies* – People live to different ages; all patterns fluctuate.
- *Individual things are unpredictable* – You cannot predict exactly how long you will live; individual points are not predictable.
- *Groups of things from a constant system of causes tend to be predictable* – Actuaries rely on this to predict life expectancy for insurance companies; a series of points from a constant process tend to follow a pattern.

It is possible to calculate statistical control limits for any given set of data and to evaluate the data against those limits. When the data fits within the limits its fluctuations are said to have a natural pattern. If the data falls outside of the limits it is said to have an unnatural pattern. The limits can either be defined in terms of a firm number, such as "centerline +/- 3 standard deviations", or in terms of pattern tests like "4 successive points greater than 2.0 standard deviations from the center line on the same side of the center line".

The primary method for such evaluation of the process is the *Control Chart*. Methods for preparation and interpretation of control charts have been developed over the last 40 years. Control Charts are employed by a wide range of industries and agencies as a means to monitor and stimulate improvements in many types of processes.

9.2 Case Study for PI Real-Time SQC

The following example illustrates how PI SQC can be used and helpful in a typical setting.

Tip: If you are not familiar with the principals and practices of Real-Time SQC, read Section 9.1, *Introduction to Statistical Quality Control* before you read this application example.

Assume that we want to set up an SQC Alarm that will evaluate data entered for a lab test. We’ve already established a PI point for the manual entry of laboratory results, and we have been entering data for some time.

Recently, we decided to apply SQC to this laboratory measurement. We used FactoryTalk Historian ProcessBook and the SQC Add-In for ProcessBook to construct an ad-hoc SQC chart of individuals for the measurement. By investigating data collected when the process was running within norms, we established control limits for the chart. We entered those numeric values into the control parameters tab for that chart. We decided to apply only the *1 of 1, Outside 3 Sigma* limit to this chart.

The ad-hoc chart served us well for a couple of months; it gave us enough information to make process improvements. After observing the post-process improvement data for some time, we recalculated control limits for this lab measurement and found that we could tighten the chart control limits. Since we decided to keep a history of our control limits over time, we created PI points into which we entered both the old and the newly calculated control limits. We then set the control parameters for the chart to get limits from the new PI tags.

After using this chart for a while, we decided that we wanted to alert the operator when the chart showed that a pattern test failure had occurred, so that action could be taken before we produced off-spec product. It wasn’t practical for the operator to keep the ad-hoc SQC chart displayed in the monitor all the time, so we decided to implement an SQC Alarm in PI.

We configured an SQC alarm point to use the same lab value as the ad-hoc chart, the same control limit tags and to evaluate the same pattern test (Individual Measurements). Now the SQC calculations happen on the PI Server whenever the lab tester enters a new value. The operator sees an indication on the FactoryTalk Historian ProcessBook display if the calculation results in an alarm condition. The operator can take corrective action and acknowledge the alarm. If the process is in an upset condition, the operator can suspend the evaluation of the alarm by clicking on a button on the FactoryTalk Historian ProcessBook display - and later turn the alarm evaluation back on when the process returns to normal.

If process engineers review the SQC control limits and enter new SQC control limits lab measurement, the PI SQC Alarm processor senses the change and begins using the new limits for the SQC Alarm.

9.3 Real-Time SQC Definitions and Terminology

The following SQC terms and phrases are used in this chapter.

Statistical Quality Control – The use of numerical methods to help keep the characteristics of a process within boundaries.

- Statistical – drawing conclusions from numbers
- Quality – the characteristics or properties of the process
- Control – keeping something within boundaries

Process – A process is a set of conditions or causes that work together to produce a result. In an industrial setting a process can be a single control loop, a unit operation, a laboratory measurement, a task performed by a single person or a team, or virtually any combination of ‘actors’ that work together to produce a result.

Fluctuating Measurements – All processes have parameters that can be measured. Repeated measurement of a process’s parameters will show fluctuations in the parameter’s value. Fluctuations can be classified into two types:

- **Natural Fluctuation** – These are small fluctuations due to the precision of measurement, the normal small random changes in the process that cannot be assigned to a cause (Non-definable Cause). Natural fluctuation may be due to minute variability in equipment, measurement imprecision, or other similar causes.
- **Unnatural Fluctuation** – These are large, abnormal fluctuations due to some external cause. The causes of unnatural fluctuations are generally definable to some outside influence. For example, an unnatural fluctuation could arise from software malfunction, field instrument failure, putting the wrong material into the product or similar causes.

Natural Pattern – A pattern of measurements of a process parameter exhibiting natural variability due to minute fluctuations in raw materials, equipment, measurement precision, etc. Obviously, natural fluctuations result in natural patterns. A natural pattern will always have the all of the following characteristics (source: *Statistical Quality Control Handbook*, 1956-1984, Western Electric Company.)

- Most of the points are near the centerline.
- A few of the points spread out and approach the control limits.
- None of the points (or at least only a very rare and occasional point) exceeds the control limits.

Conversely, unnatural patterns will lack one or more of the three characteristics listed above.

Unnatural Pattern – A pattern of measurements of a process parameter exhibiting large fluctuations due to a *definable cause*.

Tests for Unnatural Patterns – There are two basic premises for testing the naturalness of a pattern:

- Test individual points versus limits.
- Test multiple points for trend – there are several commonly accepted tests for trend recognition (e.g. eight points in a row on one side of the plot center line.)

Sample Grouping – Breaking a large collection of measurements into subgroups. Evaluating subgroup averages and ranges provides a more sensitive tool for spotting process variations with definable causes (unnatural patterns).

Example

The following example illustrates the above definitions.

Over the last couple of decades there has been an active water-sampling program in the San Francisco Bay. The graph below depicts the salinity data collected since 1989 at a single monitoring station at a depth of 1 meter:

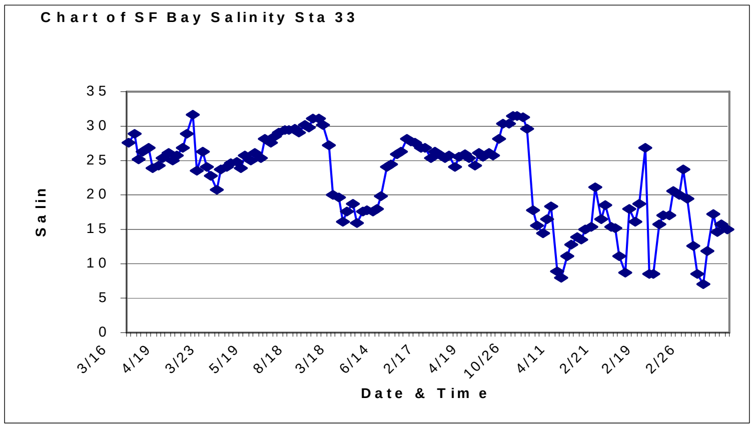


Figure 9–1. San Francisco Bay Salinity

In Figure 9–1, the process is a particular locale in San Francisco Bay. Measurements of salinity were taken over time at that location. It is clear from the plot that the measurements fluctuate. In order to learn whether the fluctuations are natural or unnatural we need to be able to test for unnatural patterns and evaluate the data against those tests.

Statistical quality control can be defined as: defining tests and evaluating data (sets of measurements) against those tests to determine if data exhibit unnatural patterns.

9.4 Tests for Unnatural Patterns

Historically, Shewart's classic Control Chart test involved testing the samples plotted on the Control Chart against 3 sigma limits. Over time, people applying SQC instituted additional tests that sought to check the samples for the presence of unnatural patterns. Over the years a number of such tests have been established. The basic set of pattern tests is known as the *Western Electric set*.

9.4.1 Western Electric Unnatural Pattern Tests

Seven classic pattern tests from the Western Electric SQC book are implemented in PI. For interpretation of patterns, the area between the upper and lower control limits is broken into 3 zones. The zones along with their names as implemented in PI SQC are illustrated below.

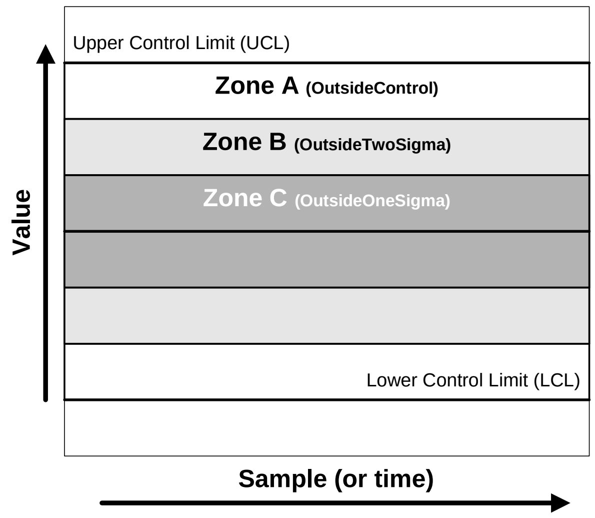


Figure 9–2. SQC Chart Zone Definition

9.4.2 Pattern Types and Tests

Instability

There are four tests for instability. These are the most important of the pattern tests:

1. Any single point that falls outside the 3-sigma limit fails this test.

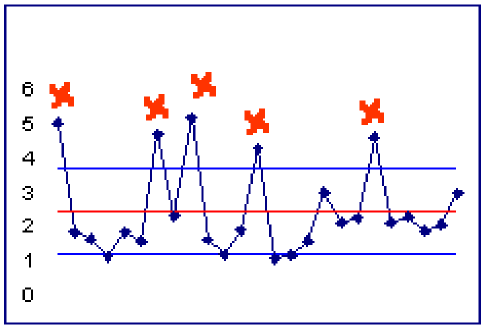


Figure 9–3. One Point Outside 3 Sigma Limit (Instability)

2. Two out of three successive points fall in Zone A or beyond on one side of the center line.

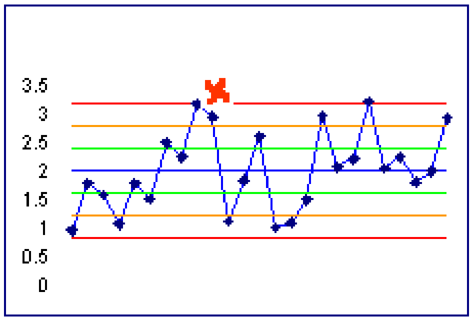


Figure 9–4. Two of Three Points in Zone A or Beyond (Instability)

3. Four out of five successive points fall into Zone B or beyond on one side of the center line.

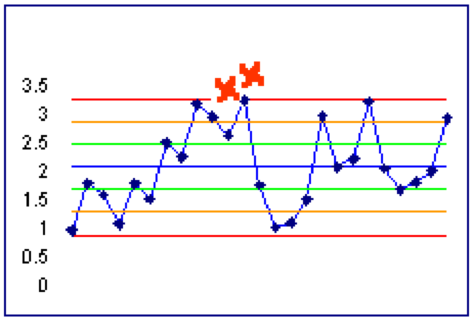


Figure 9–5. Four of Five Points in Zone B or Beyond (Instability)

4. Eight successive points fall on one side of the center line.

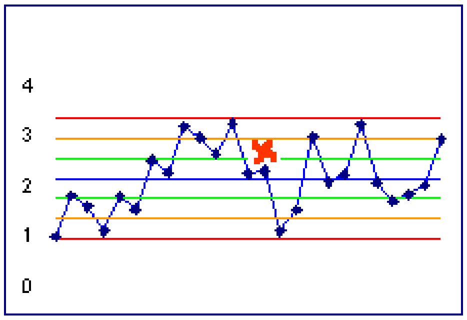


Figure 9–6. Eight Successive Points Fall on One Side of the Center Line (Instability)

Stratification

Samples whose up and down variations are very small in relation to the centerline are termed stratified. Stratification exists when 15 or more consecutive points fall within Zone C on either side of the centerline.

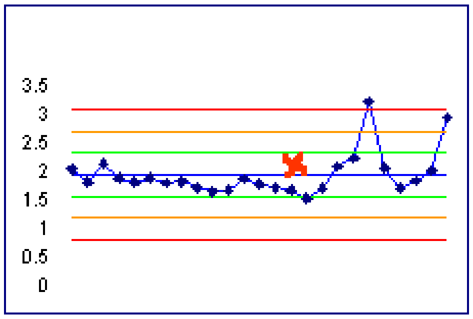


Figure 9–7. Fifteen Consecutive Points in Zone C (Stratification)

Mixtures

The mixture pattern is one in which points fall near the limits and not near the centerline. The test for mixture is eight consecutive points on both sides of the centerline where no point is within Zone C. At least one crossing of the center line must occur.

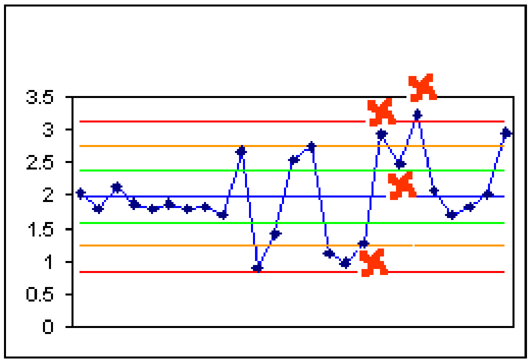


Figure 9–8. Eight Points on Both Sides of Center with None in Zone C (Mixture)

Trends

Trends are indicated by a series of points that are either monotonically increasing or decreasing.

9.5 PI Real-Time SQC Configuration

SQC Alarm processing is a critical part of implementing Real-Time SQC. The continual evaluation of SQC pattern tests and the management of alarms generated from them are the responsibility of the PI Real-Time SQC product.

Creation and maintenance of SQC Alarms is most easily performed using the **PI SQC Alarm Manager** application, which includes its own user guide. The PI point that implements PI SQC Alarm functionality is called an *SQC Alarm*. See Section 9.11, *PI SQC Alarm Point Configuration* for a detailed discussion of configuration options for an SQC Alarm.

9.5.1 Required and Optional Points

An SQC Alarm requires five points in the Data Archive in addition to the alarm point itself. Required points provide a data source for the alarm, user control over the operation of the alarm point, and storage of the SQC control limits. Required points can be shared among SQC Alarms, if appropriate.

There are five optional PI points that you can configure for an SQC Alarm. One of these optional points provides detailed reporting on the status of all pattern tests when the Alarm is set. Other optional points are used by client and user-written programs to associate specification limits and comments with an SQC Alarm and to drive limit changes based on the value of another PI point (e.g., product-based limits). See section 9.10, *Associated Point Configuration*, for point specifications.

9.6 Pattern Tests

Each PI SQC Alarm has one or more pattern tests defined. A pattern test is in Alarm status if some maximum number ***'X'*** out of a total number ***'Y'*** of consecutive samples meets the test condition (for example, if 2 of 3 samples are outside two standard deviations from the centerline). The standard Western Electric Pattern Tests are supported:

- **Outside Control** – This test counts the number of samples outside the control limit on one side of the center line.
- **Outside Two Sigma** – This test evaluates the sample against a limit drawn 2/3 of the way between the center line and the control limit. *Do not confuse sigma in this usage with the standard deviation of the sample* (that interpretation would only be true if, as classically defined, SQC control limits are set to 3 times the standard deviation of process measurements; but the control limits could be set to other values depending on process needs).
- **Outside One Sigma** – This test evaluates the sample against a limit drawn 1/3 of the way between the center line and the control limit. Do not confuse *sigma* in this usage with the standard deviation of the sample.
- **One Side of CL** – This test counts how many samples are on one side of the center line.
- **Stratification** – This test counts the number of samples that fall within the upper and lower One Sigma limits on both sides of the center line.
- **Mixture** – This test counts the number of samples that fall outside the upper and the lower One Sigma limits on both sides of the center line.
- **Trend** – This test counts the number of samples which are monotonically increasing or decreasing.

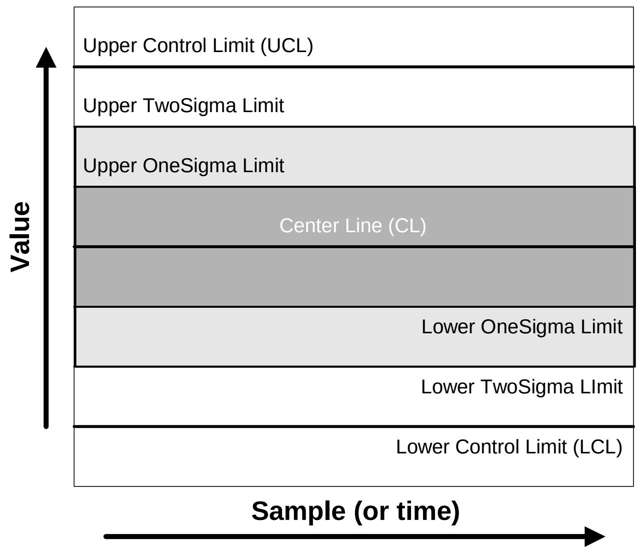


Figure 9–9. Limits for Pattern Tests

The following implementation options are supported for tests 1, 2, 3, and 4:

- Test evaluated only above the center line
- Test evaluated only below the center line
- Test evaluated both above and below the center line

9.7 SQC Alarm Priority and Precedence

The SQC Alarm processor uses the concepts of alarm priority and test precedence established in the PI Alarm Subsystem. A brief explanation appears below.

9.7.1 Priority

SQC Alarms embody the same concept of priority that is described in Chapter 8, *PI Alarm Subsystem.* Higher priority indicates a more immediate operator concern. For example, consider an alarm system designed with four priorities **0**, **1**, **2**, and **3**. A priority of **0** might indicate an alert for which no action is required, while a priority of **3** indicates a severe alarm requiring an immediate response.

Each SQC Alarm can be configured to have a priority which applies to all of the alarm's pattern tests. Selecting a priority may prove useful for incorporating SQC Alarms into *Alarm Groups* for the general management of alarms. For more information, see Section 8.6, *Build Alarm Group Points*.

9.7.2 Precedence

The seven pattern tests that may be executed with PI SQC have a fixed order of precedence, as listed below.

Table 9–1. Precedence of SQC Alarm's Pattern Tests

Precedence	Pattern Test
7	OutsideControl
6	OutsideOneSigma
5	OutsideTwoSigma
4	OneSideofCL
3	Stratification
2	Mixture
1	Trend

An SQC alarm point can be configured to test for any combination of these patterns. In such a point, if more than one pattern test is in alarm status, the highest precedence test will be the one reported in the SQC alarm point.

9.8 Create a New SQC Alarm

Follow these steps to activate a new SQC Alarm:

Caution: If you perform these steps out of order, an SQC Alarm may be placed in the Error state.

1. Start the PI Alarm Subsystem. The first time the subsystem runs, it creates the new point class used for SQC Alarms, and the Digital State Tables used for controlling SQC Alarm execution and reporting.
2. Create the Associated Points on the PI Server. You can use the Excel Workbooks that are included in the distribution as templates.
3. Enter initial values for control limit points and enter *Normal* for the **ResetTag** and the **TestStatusTag** (if used).
4. Create the SQC alarm point. You can use the Excel Workbooks included in the distribution as a template.

9.9 Start and Run the PI Alarm Subsystem

PI Real-Time SQC is handled by an enhanced version of the PI Alarm Subsystem.

❑ To start the PI Alarm Subsystem, use the command:

```
net start pialarm
```

❑ To stop the PI Alarm Subsystem, use the command:

```
net stop pialarm
```

9.9.1 Initial Subsystem Startup

The first time that the Alarm Subsystem with Real-Time SQC capability is run, it creates the **SQC_Alarm Point Class** needed to establish Real-Time SQC Alarms. The subsystem also creates the **pisqcalarm** digital state set, which contains the Alarm States used by PI SQC.

During the initial and subsequent Alarm Subsystem startups, messages are logged to the PI System Message Log, which you can review with the **pigetmsg** utility.

Determine the PointSource

When the Alarm Subsystem starts up, it determines the **PointSource** used by SQC Alarms. The default **PointSource** is **Q**. The subsystem finds all points on the server that have the specified **PointSource**.

On Windows, you can override the default **PointSource** by editing the **pialarm** key in the Registry. To do this, start RegEdit and select the key:

```
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥pialarm
```

Add a value to this key called **SQCAlarmPS**. Set the *value* equal to the letter you want for the **PointSource** (for example, **Q**).

Find all SQC Alarms

For each point found with the **PointSource** specified above, the **Scan** attribute is checked. If the **Scan** attribute is **On**, an SQC Alarm is initialized.

Initialize each SQC Alarm

After the SQC Alarm is created, its behavior is set by interpreting the configuration information in the SQC alarm point attributes. If the subsystem cannot interpret the configuration information, then a value of **Error** is written to the point and the point is not processed further. The following sections describe how SQC alarm point attributes are used during the subsystem startup.

Source Data

This is the process data source to be monitored. The **SourceTag** attribute is read and the SQC Alarm signs up for notification of new data events for the source point.

Control Limits

The **UCL**, **CL**, and **LCL Tag** attributes are read, and the SQC Alarm signs up for notification of data events on those points.

SQC Execution Control

The **ResetTag** attribute is read and the SQC Alarm signs up for notification of data events on the reset point.

TestStatusTag

The **TestStatusTag** attribute is read, and if it is a valid PI point, the SQC Alarm uses it for detailed pattern test reporting.

Pattern Tests

Each SQC Alarm contains seven pattern tests. Each pattern test is initialized by reading its configuration text.

Event Sign Up

The SQC Alarm also signs up for data events on the SQC alarm points so it will know about attempts to acknowledge alarms. The current value of the **ResetTag** is read and used to preserve the execution state of the SQC Alarm.

An SQC Alarm can be set up to clear when the subsystem restarts. If the **ClearOnStart** attribute is set to **True** then the alarm is cleared, and alarm calculations are started anew. If the **ClearOnStart** attribute is set to **False** then data are retrieved from PI to the pattern test buffer, the prior state of the SQC Alarm is retrieved from PI, and alarm calculations are restarted.

9.9.2 Subsystem Startup

Control the Evaluation of SQC Alarms

The SQC Alarm **ResetTag** attribute controls the execution state of the SQC Alarm. The three default execution states are **Normal**, **Hold**, and **Clear**:

- The **Normal** state allows the SQC Alarm to be evaluated. If **Normal** is written to the **ResetTag** of an SQC Alarm which was in **Hold**, the System Digital State **Alarm-On** (indicating that the SQC Alarm is active and awaiting a new **SourceTag** event to process) will be written to the SQC alarm point.
- The **Hold** state suspends evaluation of pattern tests. If **Hold** is written to the **ResetTag** of an SQC Alarm the System Digital State **Alarm-Off** (indicating that the SQC Alarm is no longer active) will be written to the SQC alarm point.
- The **Clear** state clears the pattern test buffer, sets the SQC Alarm in a **No Alarm State** and places the SQC Alarm in a **Hold** state, but the SQC Alarm values remains as **Clear**.

If you need to clear and restart the SQC Alarm calculations for an SQC alarm point, you must first set its **ResetTag** to **Clear** and then to **Normal**. You can do this by using the PI API in VBA code in FactoryTalk Historian ProcessBook, Microsoft Excel or in a user-written program. An example illustrating the technique appears in the FactoryTalk Historian ProcessBook independent display file, which is included in the PI SQC distribution.

Evaluate New Source Data

When a new value arrives for the **SourceTag** of an SQC Alarm, the new value is added to the collection of previous values used by the test evaluator. Each of the pattern tests is evaluated. Whenever there is a change to the alarm condition of one or more of the pattern tests, the highest precedence test in alarm is written to the SQC alarm point. The timestamp of the SQC Alarm will be the same as the source data event time.

If the optional **TestStatusTag** is configured, then a value corresponding to all tests in alarm will be written to the **TestStatusTag** with the same timestamp as the SQC Alarm event. The **TestStatusTag** also includes an indication of the side of the center line of the most recent source data event.

SQC Alarm processing is a Real-Time function; alarms are evaluated whenever new data for source points arrive. If PI Archive data are edited, or data are inserted into the Archive before the most recent Snapshot event (out of order data), then the SQC Alarm processor ignores that data.

Change the Control Limits

The control limits for an SQC Alarm can be changed at any time. When control limits are changed, new test evaluation limits are calculated for each pattern test. If the SQC Alarm's **ClearOnLimitChange** attribute of the SQC alarm point is set to **Yes**, then the SQC Alarm is cleared and alarm calculations are started anew.

If more than one limit is changed for an SQC Alarm, then more than one reset of the alarm calculations will occur. If you need to change more than one control limit, you can avoid the multiple resets by first setting the SQC Alarm's **ResetTag** to **Hold**, changing the control limits, and then setting the **ResetTag** to **Normal**. You can do this with the PI API in VBA code in FactoryTalk Historian ProcessBook, Microsoft Excel, or in a user written program. An example illustrating the technique is included in the FactoryTalk Historian ProcessBook independent display file included in the PI SQC distribution.

Control limits are applied in real-time. Once a new control limit (one that is different in value from the existing control limit) is entered, regardless of its timestamp, it is applied to the Alarms calculated from that moment in real-time forward. If a control limit value is inserted before the most recent snapshot event for the control limit, it will be ignored.

Acknowledge SQC Alarms

SQC Alarms can be configured to either of two acknowledgement options. The point attribute that controls acknowledgement behavior is **AutoAck**. The default value of the SQC Alarm's **AutoAck** attribute is **Yes**. If the default value is used, then SQC Alarms will appear just as if the SQC chart were drawn manually – each pattern test failure will result in an Alarm event. This is the typical manner for evaluating Shewart control charts.

If you wish to require personnel to manually acknowledge an SQC Alarm, set **AutoAck** to **No**. An unacknowledged alarm will be written for each alarm calculated by the pattern tests. The user has the opportunity to acknowledge the alarm by writing the unacknowledged digital state to the SQC alarm point.

9.9.3 Subsystem Shutdown

If only the Alarm Subsystem is shut down, SQC Alarm evaluation is simply terminated.

If the entire PI System is shut down, SQC Alarm evaluation is also terminated. All points that were configured to receive shutdown events will receive the shutdown value.

9.10 Associated Point Configuration

An SQC Alarm requires the existence or creation of several PI points in addition to the SQC alarm point. Also, the user has the option to create several additional PI points that could be used by user-written programs to further enhance SQC Alarm reporting. All of these points are described in the following topics, along with the requirements for configuring these points.

9.10.1 Tools to Create and Edit Associated PI Points

The SQC alarm point and its associated PI points can be configured using the PI TagConfigurator portion of PI SMT (PI System Management Tools). The PI SMT can be downloaded from the Rockwell Automation support Web site. A sample Excel workbook for use with PI TagConfigurator is included in the distribution media.

Note: When defining a new SQC Alarm, it is important that all of the required, associated PI points are created before the SQC alarm point is created.

In the following section, the function of each of the required PI points is described in detail.

9.10.2 Summary of Associated PI Points for SQC Alarms

In addition to the SQC alarm point itself, five PI points are required to implement an SQC Alarm. Additionally, five optional tags may also be created if desired.

Table 9–2. Summary of Associated Points

Tag Alias	Description
SourceTag	The PI point on which the Alarm Calculations will be performed. This point MUST have compression turned off (compressing = 0). **Note:** If compression on the source tag is not turned off, then alarms will be calculated based on snapshot events that may later be ‘compressed out.’ This may lead to confusion since alarms might be generated that are not associated with archived source-data events.
TestStatusTag	(Optional) The PI point whose value corresponds to all pattern tests currently in alarm condition.
ResetTag	The PI point whose value sets the execution status of the Pattern Test evaluation.
UCLTag	The PI point whose value is the upper control limit for the chart.
CLTag	The PI point whose value is the center line for the chart.
LCLTag	The PI point whose value is the lower control limit for the chart.
USLTag	(Optional) The PI point whose value is the Upper Specification limit for the chart.
LSLTag	(Optional) The PI point whose value is the Lower Specification limit for the chart.
ProductTag	(Optional) The PI point whose value represents the current product.
CommentTag	(Optional) The PI point for storing comments associated with the SQC Alarm.

9.10.3 Configure the Associated Points

Each of the following tag reference attributes of the SQC alarm point hold the TagName of the PI point to be used in the Alarm calculation.

Although a naming convention for the points associated with the SQC alarm point is not necessary, you may wish to implement one for ease in recognizing the purpose for the associated points. One possible convention is to base the names of the SQC alarm point and all of the associated points on the name of the SourceTag. If the source tag is *PV1011* associated tags might be named *PV1011.alarm, PV1011.status, PV1011.UCL, PV1011.LCL*, etc.

Note: For the associated points it is usually desirable to archive every value that is input for that point. Thus, exception reporting and compression are typically turned off for all associated points.

SourceTag

The **SourceTag** contains the values from which the SQC Alarm pattern tests are evaluated. This is also the value that would be charted if the user were generating a graphic SQC Chart. It is possible to use the same SourceTag for multiple SQC Alarms.

The SQC Alarm’s SourceTag can be any PI point. Typically the SourceTag point will be a manually entered value for a chart of individuals, or a PI Totalizer Point for aggregated sampled data. For example, the sampled data could be an average of a fixed number of manually entered values or a time-weighted average of an analog measurement from a DCS.

The pointtype of the SourceTag should be a numeric data type, typically a floating point number such as a float32 or float64.

To ensure that archive events for alarms and source tag archive events are in sync, the SourceTag must have compression turned off.

Note: If compression on the source tag is not turned off, then alarms will be calculated based on Snapshot events that may later be ‘compressed out.' This may lead to confusion since alarms might be generated that are not associated with archived source-data events.

TestStatusTag

The **TestStatusTag** can be used by customer-written programs in situations where knowledge of all pattern tests in alarm is needed (for example, in the dynamic creation of operator action guidelines). A TestStatusTag can only apply to one SQC Alarm.

The TestStatusTag is a point with pointtype int32, whose value is set by the Alarm Subsystem. The value indicates all of the pattern tests currently in alarm by assigning each of the pattern tests a bit within the 32-bit word. Whenever a pattern test is in alarm, its bit is set. If a test is not evaluated, its bit is always zero. After the bits are set for all pattern tests in alarm, the sign of the TestStatusTag is set to indicate whether the corresponding value of the SourceTag was above or below the center line for the event triggering the alarm. If the value of the SourceTag was equal to the center line, the TestStatusTag is positive.

The Alarm Subsystem assigns the values given in Table 9–3 to the TestStatusTag based on the individual SQC pattern-test alarms.

Table 9–3. TestStatusTag Bits Indicate SQC Alarm State

Test	Value	Bit #
OutsideControl	64	6
OutsideTwoSigma	32	5
OutsideOneSigma	16	4
OneSideofCL	8	3
Stratification	4	2
Mixture	2	1
Trend	1	0

A TestStatusTag value of **0** indicates no tests in alarm; a value of **64** indicates the **OutsideControl** test is in *alarm*; a value of **24** indicates that both the **OutsideOneSigma** and **OneSideofCL** tests are in *alarm*.

For values of the SourceTag below the centerline, the integer generated by the above bit setting algorithm is then multiplied by -1.

Table 9–4 provides more examples.

Table 9–4. Examples of TestStatusTag values

Tests in Alarm		TestStatusTag
OneSideofCL & Stratification	8 + 4	12
OutsideTwoSigma & Trend (SourceTag < CL)	32 + 1	-33
OutsideTwoSigma & OutsideOneSigma	32 + 16	48

For values of the SourceTag below the centerline, the TestStatusTag will be negative; however, care must be taken when analyzing the individual bits of the TestStatusTag. Before analyzing the bits of a negative TestStatusTag, the TestStatusTag should first be multiplied by -1 to remove the sign. Then the individual bits may be readily analyzed. For example, if the OneSideofCL alarm has been set because the values of the SourceTag are below the centerline, the TestStatusTag will first be assigned a value of 8 and then multiplied by -1 to arrive at -8. When viewed as a group of bits that are set, -8 is represented as 11111111111111111111111111111000, not -100.

ResetTag

The **ResetTag** is used by client programs to control the execution of pattern tests on an SQC Alarm. The PI Alarm Subsystem looks for updates to the value of the ResetTag and implements the desired actions. The allowed values of the ResetTag are listed in Table 9–5.

Table 9–5. Values of the ResetTag

Value	Action	Digital State Offset
Normal	Pattern Tests are processed	0
Hold	Pattern Tests are not processed, but buffer is preserved (no new values are added to it, however)	1
Clear	SQC alarm point set to the "no alarm" condition, buffer is cleared, and the SQC Alarm behavior is the same as if it were placed in Hold. (i.e. Pattern Tests are not processed. No new values are placed in the buffer.) After Clearing an alarm it is necessary to set the ResetTag to Normal to restart the processing of the SQC Alarm.	2

The same ResetTag PI point can be used for multiple SQC Alarm calculations if desired.

The PointType of the ResetTag must be Digital and the **DigitalSet** attribute should be set to **pialarmcontrol**, which is the Digital State Set where the reset values are stored. This Digital State Set is created automatically when the subsystem starts up for the first time.

UCLTag, CLTag and LCLTag

Control limits are stored in the PI points referenced by **UCLTag**, **CLTag**, and **LCLTag** parameters. These PI points can be performance equations, manually entry points, or points whose values come from a DCS or a user-written program. The SQC pattern test evaluation portion of the PI Alarm Subsystem monitors the values of the control limits and senses changes in their values (e.g., if the user implements limits that change with product or grade). The same **UCL**, **CL**, and **LCL** points can be shared among multiple SQC Alarms if desired.

The pointtype of the control limit points should be set a numeric data type, such as a float32 or float64.

If a control limit is changed, the pattern tests for the associated SQC Alarm are reset, depending on the **ClearOnLimitChange** attribute. If **ClearOnLimitChange = Yes**, the SQC Alarm is placed in the **Hold** state, the **TestBuffer** is zeroed and the alarm evaluation is then set to the **Normal** state.

USLTag and LSLTag

Specification limits are stored in the PI points referenced by the optional **USLTag**, and **LSLTag** attributes. These PI points can be performance equations, manually entry points, or points whose values come from a DCS or a user-written program.

The pointtype of the control limit points should be set to a numeric data type, such as a float32 or float64.

These tags are used for display and for calculation of the process capability (Cpk) by version 1.1 or greater of the PI SQC Add-In for ProcessBook. If the attributes are blank, then the specification limits will not be displayed and the Cpk will not be calculated.

While specification limits are not used in SQC Alarm calculations, users may wish to establish regular PI Alarms based on the values of the SourceTag relative to the specification points.

ProductTag

ProductTag is an optional PI point used to designate the current product for which the SQC Alarm is being calculated. User-written programs might use this point's value to programmatically adjust the control and specification limits based on the product or grade of material being manufactured.

CommentTag

CommentTag is an optional PI point for storing comments associated with the SQC Alarm. Information regarding conditions in the plant environment at certain times may be stored in this tag. User-written programs may be used to enter and retrieve these comments for use in process improvements based on attributing SQC Alarms to causes such as the technique known as Pareto analysis.

9.11 PI SQC Alarm Point Configuration

The **PI SQC Alarm Point** is the central storage point for all the information needed to implement a PI SQC Alarm. Configuration information of a SQC alarm point includes all of the attributes needed to define the PI SQC Alarm’s behavior as well as the associated points that are referenced by the SQC Alarm.

The following topics describe how to manually configure the PI SQC alarm point. OSI recommends the use of the **PI SQC Alarm Manager** application for the creation and maintenance of SQC Alarms.

Later, you will find discussion of how to configure associated points, both required and optional, that are used to define a SQC Alarm.

9.11.1 Methods for Configuring PI SQC Alarm Points

The order of point creation is important, because if the SQC alarm point is created before the associated points (including setting their initial values), then the PI Alarm Subsystem will not process the SQC Alarm; instead, it will place the SQC Alarm in the **Error** state.

The PI SQC Alarm Manager is a utility included in the distribution media that will enable you to create and edit alarm points correctly.

Alarm points can also be configured using the PI TagConfigurator portion of PI-SMT (PI System Management Tools). PI TagConfigurator currently supports creating, editing, and deleting SQC Alarm tags. Sample Excel workbooks for use with PI TagConfigurator are included in the distribution media. The instructions in the workbooks step through the creation of associated points, setting initial values for the associated points, creating the sample aggregation point (for SQC Alarms other than for a chart of Individuals), and the creation of the SQC alarm point itself.

9.11.2 SQC Alarm Point Class

SQC Alarm Definition

SQC alarm points have their own point class: **SQC_Alarm**. This **SQC_Alarm** point class contains attributes needed to describe an SQC Alarm. The fundamental attributes that are required to specify an SQC alarm point are given in Table 9–6.

Note: When specifying attribute values 'Yes', 'Y', 'True', 'T', 'On' and '1' all have equivalent meaning and may be used interchangeably. The strings are case insensitive. The same is true of 'No', 'N', 'False', 'F', 'Off' and '0'.

Table 9–6. SQC_Alarm Point Class Attributes

Point Attribute	Default Value or Valid Options	Description
Ptclass	SQC_Alarm	The SQC Alarm Point Class
PointType	Digital	All SQC Alarms are Digital points
Digitalset	pisqcalarm	Contains SQC Alarm States
PointSource	Q	Identifies SQC alarm points
Scan	1 or 0 (default=1)	On (1) or Off (0). SQC Alarms are only calculated for scan=On
Compressing	1 or 0 (default=1)	On or Off. Should be set to OFF
AutoAck	Yes (default) or No	Automatic acknowledgement of alarms
ChartType	0-8 (0 is default)	Describes type of SQC chart the SQC Alarm is testing (Used only by the PI SQC Add-In for ProcessBook).
ClearOnStart	Yes or No (default)	Yes means start alarm calculations afresh on startup or editing of the alarm point; No means seed alarm calculations with archive values of SourceTag and restore the alarm's prior state
ClearOnLimitChange	Yes (default) or No	Yes means start alarm calculations afresh when any limit tag changes; No means continue evaluating alarms using new limit values.
ExDesc		Specifies Alarm Group

Point Attribute	Default Value or Valid Options	Description
SQCAlarmPriority	0-n (0 is default)	Integer value specifies the alarm priority assigned to all alarms generated by this SQC Alarm definition.
PIProductLimits		<reserved for future use>
WaitOnLimitChange		<reserved for future use>
Pattern Test Definitions (if left blank, the pattern test will not be performed)		
OutsideControl	x of y [<blank>, above, or below]	Within y number of samples, x are outside of control limits. Options: specify above or below to apply test only above or below the center line.
OutsideTwoSigma	x of y [<blank>, above, or below]	Within y number of samples, x are outside the "Two Sigma" limit. Options: specify above or below to report alarms for this test only above or below the center line.
OutsideOneSigma	x of y [<blank>, above, or below]	Within y number of samples, x are outside the "One Sigma" limit. Options: specify above or below to report alarms for this test only above or below the center line.
OneSideofCL	x of y [<blank>, above, or below]	Within y number of samples, x are on one side of the center line. Options: specify above or below to report alarms for this test only above or below the center line.
Stratification	x of y	Within y number of samples, x are within the "One Sigma" limit.
Mixture	x of y	Within y number of samples, x are not within the "One Sigma" limit.
Trend	x	x consecutive values trend either up or down.
Associated-Point TagName Definitions		
SourceTag		TagName for PI point on which the Alarm Calculations will be performed
TestStatusTag		(Optional) TagName for PI point to which SQC Alarm system writes value indicating which tests are in alarm
UCLTag		TagName for Upper Control Limit
CLTag		TagName for Center Line

Point Attribute	Default Value or Valid Options	Description
LCLTag		TagName for Lower Control Limit
ResetTag		TagName for PI point governing Alarm Calculation execution and reset
USLTag		(Optional) TagName for Upper Specification Limit
LSLTag		(Optional) TagName for Lower Specification Limit
ProductTag		(Optional) TagName for a PI point to store the name of the current product or grade being produced.
CommentTag		(Optional) TagName for a PI point to store comments associated with the SQC Alarm.

Ptclass

This must be set to SQC_Alarm.

PointType

This must be set to digital. The SQC alarm point contains a digital state representing the value of the highest precedence alarm currently set.

Digitalset

This must be set to **pisqcalarm**, which contains SQC Alarm States. The digital states in the **pisqcalarm** Digital State Set are described in "Default SQC Alarm Digital States," page 376.

PointSource

The **PointSource** identifies points as SQC alarm points. The default **PointSource** for SQC Alarms is **Q**, although this can be configured to a different string as described in the topic, *Determine the PointSource,* in section 9.9.1.

Scan

Scan is used to determine when SQC Alarms are calculated. SQC Alarms are not calculated when **Scan = Off.** When **Scan** is changed from **Off** to **On** while the SQC Alarms are running, the **ClearOnStart** attribute (described below) determines the SQC Alarm's startup behavior.

Compressing

Compression defaults to **On** (i.e., 1) for all PI points, but **Compressing** should be set to **OFF** (i.e., **0**) to ensure archiving of all SQC Alarms.

Note: Exception reporting is not used for SQC alarm points.

Behavior Controls

AutoAck

AutoAck is used for automatic acknowledgement of alarms. It defaults to **Yes**.

If **AutoAck = No**, the **SQCAlarmPriority** needs to be set to a nonzero value in order to use acknowledgements.

ChartType

ChartType is an indicator used by the PI SQC Add-In to FactoryTalk Historian ProcessBook. The **ChartType** attribute indicates the type of SQC chart for which the SQC Alarms are being calculated. Valid values are listed in Table 9–7.

Note: This attribute is not used by the PI Server in its SQC calculations. The SQC Alarm Point's SourceTag will determine whether this is a chart of individuals, an x-bar chart, etc. The default value is **0**. See page 375 for a description of how to set up SQC Alarms for the various chart types.

Table 9–7. Valid ChartType Values

Value	ChartType
0	Chart type is unspecified (default)
1	Individuals
2	X-Bar
3	Moving Average
4	Exponentially-Weighted Moving Average (EWMA)
5	Standard Deviation
6	Moving Standard Deviation
7	Range
8	Moving Range

ClearOnStart

ClearOnStart=Yes is used to clear any active alarm and start alarm calculations afresh on startup or after a change to the alarm point's attributes. No means to start the alarm calculations using retrieved archive values of SourceTag and to restore the SQC Alarm's prior state. If not specified on point creation, **ClearOnStart** defaults to **No**.

ClearOnLimitChange

ClearOnLimitChange=Yes means clear any active alarm and start alarm calculations afresh when any control limit tag (**UCLTag**, **CLTag**, **LCLTag**) changes. **No** means continue evaluating alarms using new limit values. If not specified on point creation, **ClearOnStart** defaults to **Yes**.

Note: If set to **Yes**, changing more than one limit tag may result in one reset for each limit that is changed. This can be avoided by first setting the **ResetTag** to **Clear**, changing the control limits, and then setting the **ResetTag** to **Normal**.

PIProductLimits

<Included in Point class and reserved for future use>

WaitOnLimitChange

<Included in Point class and reserved for future use>

Alarm Group Identification

ExDesc

ExDesc specifies an Alarm Group that this SQC Alarm belongs to. SQC Alarms may be added to existing PI Alarm Groups.

Alarm Priority

SQCAlarmPriority

SQCAlarmPriority is an integer value that sets the priority of the SQC Alarm. Defaults to **0**. This affects how the SQC Alarm will be reported with respect to other alarms and SQC Alarms on your system. For a discussion of alarm priorities and precedence, see page 356.

In addition to setting **AutoAck = No**, SQCAlarmPriority must be set to a nonzero value in order to use acknowledgements.

9.11.3 Pattern Test Configuration

Each SQC Alarm can test for up to seven pre-set patterns. Any combination of pattern tests may be used in a single alarm. Pattern tests that compare values of the source tag to the control limits use values of the **UCLTag**, **CLTag**, and **LCLTag**. A description of each test follows below.

The general form for configuring a pattern test is to enter the text "**x of y**", where **x** and **y** are positive integers. If a pattern test attribute is left blank it will not be evaluated. The descriptions of each pattern test also list the values for **x of y** recommended in the *Western Electric Statistical Quality Control Handbook.*

In the case of the Trend pattern test only, **x** is specified because the test looks for **x** consecutively increasing or decreasing values.

For four of the tests (**OutsideControl**, **OutsideOneSigma**, **OutsideTwoSigma**, and **OneSideofCL**), the modifiers above and below can be added after "**x of y**" to restrict the alarm reporting to a single side of the center line. Table 9–8 illustrates the pattern test configuration options.

Table 9–8. Pattern Test Configuration Examples

Attribute Entry	Pattern Test Behavior
<Blank>	Test will not be performed (default).
4 of 5	Test looks for 4 out of 5 consecutive values of the SourceTag which meet the test condition. This test will be evaluated for values of the SourceTag that are both greater than and less than the value of the CLTag
2 of 3 above	Test will look for 2 out of 3 consecutive values of the SourceTag which meet the test condition. Alarms resulting from this test will only be reported for values of the SourceTag that are greater than the value of the CLTag.
8 of 8 below	Test will look for 8 consecutive values of the SourceTag which meet the test condition. Alarms resulting from this test will only be reported for values of the SourceTag that are less than the value of the CLTag.

OutsideControl

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are outside of control limits. A value of the SourceTag is outside the control limits when

```
SourceTag > UCL or SourceTag < LCL.
```

The above or below options can be used with this pattern test to restrict pattern alarm reporting to one side of the center line. The Western Electric SQC Handbook recommends 1 of 1 for this pattern test.

OutsideTwoSigma

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are outside the "TwoSigma" limit. A value of the SourceTag is outside the "TwoSigma" limit when

```
SourceTag > (2/3 * (UCL-CL) + CL) or SourceTag < (CL - 2/3 * (CL-LCL)).
```

The above or below options can be used with this pattern test to restrict pattern alarm reporting to one side of the center line. The Western Electric SQC Handbook recommends 2 of 3 for this pattern test.

OutsideOneSigma

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are outside the "OneSigma" limit. A value of the SourceTag is outside the "OneSigma" limit when

```
SourceTag > (1/3 * (UCL-CL) + CL) or SourceTag < (CL - 1/3 * (CL-LCL)).
```

The above or below options can be used with this pattern test to restrict pattern alarm reporting to one side of the center line. The Western Electric SQC Handbook recommends 4 of 5 for this pattern test.

OneSideofCL

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are on one side of the center line. The above or below options can be used with this pattern test to restrict pattern alarm reporting to one side of the center line. The Western Electric SQC Handbook recommends 8 of 8 for this pattern test.

Stratification

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are within the "OneSigma" limit. A value of the SourceTag is within the "OneSigma" limit when

```
(CL - 1/3 * (UCL-CL)) > SourceTag > (CL + 1/3 * (CL-LCL)).
```

The Western Electric SQC Handbook recommends 15 of 15 for this pattern test.

Mixture

This pattern test fails (sets an alarm) when **x of y** values of the SourceTag are found on both sides of the centerline and none of these values falls within "OneSigma." Each of the x values must satisfy one of the following two conditions

```
SourceTag > (CL + 1/3 * (UCL-CL)) or SourceTag < (CL - 1/3 * (CL-LCL));
```

and among the **x** values, both of these conditions must be met at least once.

The Western Electric SQC Handbook recommends 8 of 8 for this pattern test.

Trend

This pattern test fails (sets an alarm) when **x** consecutive values of the SourceTag trend either up or down.

The Western Electric SQC Handbook recommends 8 for this pattern test.

9.11.4 Associated-Point Tagnames

There are several required and optional tags associated with each SQC Alarm. The tag names of these points are stored in the attributes described below.

SourceTag

Tagname for PI point containing the source data on which the SQC Alarm calculations will be performed.

TestStatusTag

(Optional) Tagname for PI point to which SQC Alarm system writes value indicating which tests are in alarm. When this attribute is left blank (the default) the TestStatusTag is not used. When a TestStatusTag is used, the TestStatusTag point must be unique in each SQC Alarm definition.

UCLTag

Tagname for Upper Control Limit. The same UCLTag can be used in more than one SQC Alarm definition.

CLTag

Tagname for Center Line. The same CLTag can be used in more than one SQC Alarm definition.

LCLTag

Tagname for Lower Control Limit. The same LCLTag can be used in more than one SQC Alarm definition.

ResetTag

Tagname for PI point governing the SQC Alarm calculation's execution and reset. The same ResetTag can be used in more than one SQC Alarm definition.

USLTag and LSLTag

(Optional) Tagnames for the PI points to store the Upper Specification and Lower Specification Limits. The same USLTag and LSLTag can be used in more than one SQC Alarm definition. Leave these tag attributes blank (default) if you do not wish to store your specification limits in PI points.

ProductTag

Reserved for future use

CommentTag

(Optional) Tagname for a PI point to store comments associated with the SQC Alarm. Leave this tag attribute blank (default) if you do not wish to store comments in a PI point.

9.12 PI Real-Time SQC Chart Types

The type of control chart for which an SQC Alarm is established depends solely on the SQC Alarm’s SourceTag. The SourceTag can be any PI point type; creating subgroups of measurements for use in SQC is generally done using a combination of totalizer and performance equation points.

9.12.1 Charts of Individuals

The SourceTag for the SQC Alarm here is any PI point. Each new measurement is used in the pattern test evaluation

9.12.2 Moving Average, Moving Range and Moving Standard Deviation

The SourceTag for these types of SQC Alarms is a Totalizer point whose source is the raw measurement point. The Totalizer point should configured to take the moving average, etc. of the raw data point based either on a number of raw data events or a time period.

9.12.3 X-Bar, Range and Standard Deviation

The SourceTag for these types of SQC Alarms is a Totalizer point whose source is the raw measurement point. The Totalizer point should configured to take the average, etc. of the raw data point based either on a number of raw data events (**nsampleblock**) or a time period.

9.12.4 EWMA

The SourceTag for the EWMA (Exponentially Weighted Moving Average) SQC Alarm is a Performance Equation which watches for events on the raw data tag and whose equation syntax is (note that <xxx> indicates that you need to substitute a value or string):

```
 ‘<RawDataTag>’  +
if badval ( ‘<me>’ ) then
<Lambda> * PrevVal(<’ RawDataTag>’ ) else
<Lambda> * PrevVal( ‘<me>’ )
```

Lambda is the weighting factor whose magnitude you must determine. You should substitute the name of the Performance Equation tag that you are creating for <me> in the example above.

9.13 Default SQC Alarm Digital States

The default digital state set (**pisqcalarm**) for SQC Alarms is automatically installed during the first startup of the PI Alarm Subsystem. Another Digital State Set can be created for custom use. The Digital State Set must contain the same seven pattern tests in the same order as below. You may alter the number of acknowledgement states or priorities or change the text displayed when a pattern test fails in your custom set. See Create a New SQC Alarm for details on constructing Alarm Digital State Sets.

The zeroeth state contains the **No Alarm** digital state. The last digital state (71st in this case), records the number of priorities and acknowledgement states in the digital state set.

Table 9–9. Default DigitalSet for SQC Alarms

Offset	AlarmSet
0	.
1	__ Trend
2	_* Trend
3	** Trend
4	__ Trend <<
5	_* Trend <<

Offset	**AlarmSet**
6	** Trend <<
7	__ Trend _x
8	_* Trend _x
9	** Trend _x
10	__ Mixture
11	_* Mixture
12	** Mixture
13	__ Mixture <<
14	_* Mixture <<
15	** Mixture <<
16	__ Mixture _x
17	_* Mixture _x
18	** Mixture _x
19	__ Stratification
20	_* Stratification
21	** Stratification
22	__ Stratification <<
23	_* Stratification <<
24	** Stratification <<
25	__ Stratification _x
26	_* Stratification _x
27	** Stratification _x
28	__ OneSideofCL
29	_* OneSideofCL
30	** OneSideofCL
31	__ OneSideofCL <<
32	_* OneSideofCL <<
33	** OneSideofCL <<
34	__ OneSideofCL _x
35	_* OneSideofCL _x
36	** OneSideofCL _x
37	__ OutsideTwoSigma
38	_* OutsideTwoSigma

Offset	AlarmSet
39	** OutsideTwoSigma
40	__ OutsideTwoSigma <<
41	_* OutsideTwoSigma <<
42	** OutsideTwoSigma <<
43	__ OutsideTwoSigma _x
44	_* OutsideTwoSigma _x
45	** OutsideTwoSigma _x
46	__ OutsideOneSigma
47	_* OutsideOneSigma
48	** OutsideOneSigma
49	__ OutsideOneSigma <<
50	_* OutsideOneSigma <<
51	** OutsideOneSigma <<
52	__ OutsideOneSigma _x
53	_* OutsideOneSigma _x
54	** OutsideOneSigma _x
55	__ OutsideControl
56	_* OutsideControl
57	** OutsideControl
58	__ OutsideControl <<
59	_* OutsideControl <<
60	** OutsideControl <<
61	__ OutsideControl _x
62	_* OutsideControl _x
63	** OutsideControl _x
64	Trend
65	Mixture
66	Stratification
67	OneSideofCL
68	OutsideTwoSigma
69	OutsideOneSigma
70	OutsideControl
71	3 3

9.14 Log Messages

Errors in configuration of SQC alarm points and errors encountered in the operation of the subsystem are written to the PI System Message Log. The following section explains how to view those error messages using the **PIGetMsg** utility. The error messages are listed along with hints on how to correct the error condition.

9.14.1 View Log Messages

The **PIGetMsg** utility provides the means to view all SQC-related messages written to the PI System Message Log. Use the following command to view them:

```
pigetmsg -st "pitimestring" -et "pitimestring" -pn "pialarm" -msg "mask"
```

Substitute a valid time string in PI format for the *start* and *end pitimestring* and substitute the string mask you are interested in for mask. The mask could be a tagname.

9.14.2 Log Message Reference

Wherever you see <SQC Alarm Point> in this list, you will see the name of your SQC alarm point within the braces in the message log.

Table 9–10 lists the messages that occur under normal operating conditions.

Table 9–10. Informational Messages

Message	Explanation
Created the state set	The subsystem created one of the digital state sets it needs to function. Only seen if the state set does not exist at the time of subsystem startup.
SQC alarm point class OK	The subsystem created the SQC Alarm pointclass. Only seen if the pointclass does not exist when the subsystem starts up. The subsystem successfully created the pointclass.
Adding SQC Alarm <SQC Alarm Point>	A new SQC alarm point has been created on the PI Server and it is being picked up by the subsystem.
Previously deleted SQC Alarm <SQC Alarm Point> is being added again.	The SQC Alarm was previously established. PI point was deleted and then re-created. Now it is being picked up by the subsystem.
Editing SQC Alarm <SQC Alarm Point>	The SQC alarm point has been edited and the subsystem is picking up the changes.
An existing PI point is being changed to an SQC Alarm <SQC Alarm Point>	The PI point's PointSource was edited to the one used by the subsystem for SQC Alarms and is being picked up by the subsystem.
PointSource edit, deleting alarm <SQC Alarm Point>	The PI point's PointSource was edited to one not used by the subsystem, and the subsystem will no longer process the point.
PI Point Deleted, deleting alarm <SQC Alarm Point>	The PI point was deleted so the subsystem will no longer process it.

Message	Explanation
<SQC Alarm Point> Scan set to off	The SQC Alarm's **Scan** attribute was set to 'OFF', so the subsystem will no longer process it.
<SQC Alarm Point> Scan-off at initialization – not added	The subsystem tried to establish the SQC Alarm, but the point's scan attribute is set to 'OFF'. If this is not what you want, edit the point using PI SMT and change the scan attribute to 'ON'.
<SQC Alarm Point> Scan-on ... re-initializing.	SQC alarm point was edited and the **Scan** parameter was changed from 'OFF' to 'ON'. The Alarm is being put back on line.

Error Messages

Table 9–11 lists messages that indicate that a serious error has occurred at some point in the running of PI Real-Time SQC or in the initialization of a Real-Time SQC Alarm.

Table 9–11. Error Messages: Serious Errors

Message	Explanation
Failed to create SQC alarm point class ... retrying	The subsystem had trouble creating the pointclass. Possible solution to problem: Open a command window and change directory to the *pi\adm* directory. Then issue the command `net stop pialarm`. After the services are stopped, enter the command `net start pialarm`. Now look in the message log for the message '*SQC alarm point class created*'.
<SQC Alarm Point> failed to setup digital set	The digital set specified for the SQC Alarm is not valid. Check to see that you have used a valid digital state set and that it conforms to the requirements as listed in this chapter.
<SQC Alarm Point> being edited but unable to retrieve attributes	Subsystem can't get attributes for point during an attempt to edit the SQC Alarm.
<SQC Alarm Point> is wrong point type for alarm point	The SQC Alarm was not created as a digital point. Change the SQC alarm point's pointtype to digital.
<SQC Alarm Point> failed to get sourcetag attribute.Status-	Unable to retrieve the SQC Alarm's SourceTag attribute.
<SQC Alarm Point> update signup for <_sourcetag > failed	For some reason the subsystem was not able to sign up for data events on the tag.
<SQC Alarm Point> sourcetag <_sourcetag> pointtype is not valid	The SourceTag was not a recognized type, change the SourceTag pointtype.
<SQC Alarm Point> failed to get teststatustag attribute	Unable to retrieve the **TestStatusTag** attribute.
<SQC Alarm Point> failed to get SQCAlarmPriority attribute	Unable to get the **SQCAlarmPriority** attribute.

Message	Explanation
<SQC Alarm Point> update signup for <SQC Alarm Point> failed	Unable to sign up for data events on the SQC alarm point
<SQC Alarm Point> Status of < ResetTag > is bad at start	The current value of the ResetTag for the SQC Alarm is not NORMAL, HOLD or CLEAR. Enter an initial value of NORMAL for the reset tag.
<SQC Alarm Point> Status of one or more Limits is bad at start	One or more limit points for this SQC Alarm has a bad status (includes 'Pt Created', 'Shutdown', etc). Enter initial values for all three limit tags.
<SQC Alarm Point> One or more pattern tests failed to initialize	One or more of the pattern test attributes was not properly defined. Check to see that all pattern tests are properly configured according to this chapter.

TECHNICAL SUPPORT AND RESOURCES

Technical Support Options

Contact Rockwell Automation Technical Support at the following:

- Customer Support Telephone — 1-440-646-3434
- Online Support — **http://support.rockwellautomation.com**

Knowledge Center

The Knowledge Center provides a searchable library of documentation and technical data, as well as a special collection of resources for system managers. For these options, click **Knowledge Center** in the Technical Support Web site.

- The **Search** feature allows you to search Support Solutions, Bulletins, Support Pages, Known Issues, Enhancements, and Documentation (including User Manuals, Release Notes, and White Papers).
- **System Manager Resources** include tools and instructions that help you manage: Archive sizing, Backup scripts, Daily Health Check, Daylight Saving Time configuration, PI Server security, PI System sizing and configuration, PI Trusts for Interface Nodes, and more.

Before You Call or Write for Help

When you contact Rockwell Automation Technical Support, please provide:

- Product name, version, and/or build numbers
- Computer platform (CPU type, operating system, and version number)
- The time that the difficulty started
- The message log(s) at that time

Find Version and Build Numbers

To find version and build numbers for each PI System subsystem (which vary depending on installed upgrades, updates or patches) use either of the following methods:

- If you have PI System Management Tools (PI SMT) installed, select **Start > Programs > PI System > PI System Management Tools**. In PI SMT, select the server name, then under *System Management Plug-Ins*, open **Operation > PI Version**. The PI Version tree lists all versions.

- If you do not have PI SMT installed, open a command prompt, change to the *pi\adm* directory, and enter `piversion -v`. To see individual version numbers for each subsystem, change to the *pi\bin* directory and type the subsystem name followed by the option `-v` (for example, `piarchss.exe -v`).

View Computer Platform Information

To view platform specifications:

- In Windows, right-click on **My Computer** and choose **Properties**. For more detailed information, select **Start** > **Run**, and enter `msinfo32.exe`
- In UNIX, enter `uname -a`

INDEX OF TOPICS