PI Interface for Rockwell FactoryTalk Batch

Version 3.0.6.x

OSIsoft, LLC

777 Davis St., Suite 250
San Leandro, CA 94577 USA

Tel: (01) 510-297-5800
Fax: (01) 510-357-8136
Web: http://www.osisoft.com

OSIsoft Australia • Perth, Australia

OSIsoft Europe GmbH • Frankfurt, Germany

OSIsoft Asia Pte Ltd. • Singapore

OSIsoft Canada ULC • Montreal & Calgary, Canada

OSIsoft, LLC Representative Office • Shanghai, People's Republic of China

OSIsoft Japan KK • Tokyo, Japan

OSIsoft Mexico S. De R.L. De C.V. • Mexico City, Mexico

OSIsoft do Brasil Sistemas Ltda. • Sao Paulo, Brazil

OSIsoft France EURL • Paris, France

Published: 07/2013

Table of Contents

Terminology

To understand this interface, familiarize yourself with the terminology used in this manual.

Interface Node

A computer on which the PI API, the PI SDK, or both are installed, and PI Server programs are not installed.

PI API

A library of functions that enable applications to communicate and exchange data with the PI Server.

PI Collective

Two or more replicated PI Servers that collect data concurrently. Collectives are part of the High Availability environment. When the primary PI Server in a collective becomes unavailable, a secondary collective member node seamlessly continues to collect and provide data access to your PI clients.

PIHOME

The directory that is the common location for PI 32-bit client applications. The [PIHOME] directory tree is defined by the PIHOME entry in the pipc.ini configuration file. This pipc.ini file is an ASCII text file, which is located in the %windir% directory.

On a 32-bit operating system a typical PIHOME is C:\Program Files\PIPC.

On a 64-bit operating system a typical PIHOME is C:\Program Files (x86)\PIPC.

PI interfaces reside in a subdirectory of the Interfaces directory under PIHOME.

For example, files for the Modbus Ethernet Interface are in [PIHOME]\PIPC\Interfaces\ModbusE.

This document uses [PIHOME] as an abbreviation for the complete PIHOME or PIHOME64 directory. For example, ICU files in [PIHOME]\ICU.

PIHOME64

The directory that is the common location for PI 64-bit client applications on a 64-bit operating system.

A typical PIHOME64 is C:\Program Files\PIPC.

PI interfaces reside in a subdirectory of the Interfaces directory under PIHOME64.

For example, files for a 64-bit Modbus Ethernet Interface are found in

C:\Program Files\PIPC\Interfaces\ModbusE.

This document uses [PIHOME] as an abbreviation for the complete PIHOME or PIHOME64 directory. For example, ICU files in [PIHOME]\ICU.

PI SDK

A library of functions that enable applications to communicate and exchange data with the PI Server. Some PI interfaces, in addition to using the PI API, require the PI SDK.

AF SDK

A library of functions that enable applications to communicate and to exchange data with the AF Server. Some PI interfaces, in addition to using the PI API, PI-SDK, require the AF SDK.

PI Server Node

A computer on which PI Server programs are installed. The PI Server runs on the PI Server Node.

PI SMT

PI System Management Tools. PI SMT is the program you use for configuring PI Servers. A single copy of PI SMT manages multiple PI Servers. PI SMT runs on either a PI Server Node or a PI Interface Node.

Pipc.log

The file to which OSIsoft applications write informational and error messages. While a PI interface runs, it writes to the pipc.log file. The ICU provides easy access to the pipc.log.

Point

The basic building block for controlling data flow to and from the PI Server. For a given timestamp, a PI point holds a single value.

A PI point does not necessarily correspond to a "data collection point" on the foreign device. For example, a single "point" on the foreign device can consist of a set point, a process value, an alarm limit, and a discrete value. These four pieces of information require four separate PI points.

Service

A Windows program that runs without user interaction. A Service continues to run after you have logged off as a Windows user. A Service has the ability to start up when the computer itself starts up.

The ICU enables you to configure a PI interface to run as a Service.

Tag (Input Tag and Output Tag)

The name of the PI point. There is a one-to-one correspondence between the name of a point and the point itself. Because of this relationship, PI System documentation uses the terms "tag" and "point" interchangeably.

Interfaces read values from a device and write these values to an Input Tag. Interfaces use an Output Tag to write a value to the device.

Chapter 1. Introduction

This manual describes the operation of the PI Interface for Rockwell FactoryTalk Batch. The primary objective of the Batch Interface is to collect batch data from the Rockwell FactoryTalk system through Event Journals (EVT files) and store them in the PI Batch Database or PI AF Database (as event frames). In addition to collecting batch data, the interface collects associated batch data to PI Tags and PI Batch properties.

The flow of data in the interface is unidirectional: data can only be read from the specified data source and written to the PI Server. This interface can read data from multiple batch data sources simultaneously. By design, the interface does not edit or delete source data.

The Batch Interface is a scan-based interface that populates the AF Database (event frames and elements) or the PI Batch Database and PI Module Database. In addition to batch data, the interface can populate the PI Point Database. PI Point creation, commonly known as tag creation and event population, is controlled by using tag templates. All modules, tags, tag aliases, and health tags are automatically created on the PI server. The Interface does not use the PI API Buffering Service, because batch and tag data is already buffered by the source historian databases. To maximize performance, the interface writes events to PI tags in bulk—that is, it writes all events per interface scan.

Reference Manuals

OSIsoft

- PI Data Archive Manual
- PI Server System Management Guide
- PI SDK User Manual

Vendor

Review the pertinent documentation regarding the particular Batch Executive System (BES) at your facility. Maintain familiarity with the contents and format of the source data so that you can choose appropriate options and features for the interface.

Supported Features

Feature	Support	
Part Number	PI-IN-RW-FTB-NTI	
* Platforms	32-bit Interface	64-bit Interface
Windows XP		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows 2003 Server		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows Vista		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Windows 2008		
32-bit OS	Yes	No
Windows 2008 R2		
64-bit OS	Yes (Emulation Mode)	No
Windows 7		
32-bit OS	Yes	No
64-bit OS	Yes (Emulation Mode)	No
Auto Creates PI Points	Yes	
Point Builder Utility	No	
ICU Control	No. Note: To configure the interface, use PI Event Frames Interface Manager (included)	
PI Point Types	Integer / Float32 / String	
Sub-second Timestamps	Yes	
Sub-second Scan Classes	No	
Automatically Incorporates PI Point Attribute Changes	No	
Exception Reporting	No	
Outputs from PI	No	
Inputs to PI	Event and Scan-based	
Supports Questionable Bit	No	
Supports Multi-character PointSource	Yes	
Maximum Point Count	None	
* Uses PI SDK	Yes	
PINet String Support	N/A	

Feature	Support
* Source of Timestamps	Device
History Recovery	Yes
UniInt-based * Disconnected Startup * SetDeviceStatus	No No Yes
Failover	Yes
* Vendor Software Required on PI Interface Node/PINet Node	Yes
* Vendor Software Required on Foreign Device	Yes
* Vendor Hardware Required	No
Additional PI Software Included with Interface	No
Device Point Types	String/Integer/Float
Serial-Based Interface	No

*See paragraphs below for further explanation.

Platforms

The Interface runs on the above mentioned Microsoft Windows operating systems. Newer platforms might not be supported. Please contact OSIsoft Technical Support for more information.

PI SDK

The PI SDK and the PI API are bundled and must be installed on each PI Interface node. The PI Interface for Rockwell FactoryTalk Batch makes PI SDK calls to access the PI Module Database and PI Batch Database. The Interface requires PI SDK version 1.3.4.333 or higher to be installed. The Interface uses PI API to log messages in the local pipc.log file. It does not require a PI API connection to the PI Server.

AF SDK

The AF SDK must be installed on each PI Interface node. The interface makes AF SDK calls to access AF elements and AF event frames. The interface requires AF SDK version 2.5.x or higher to be installed prior the execution of the interface.

Source of Timestamps

The timestamp accompanying the record is used as the source of the timestamp for the data to be placed into the PI system. For the health tags, the Interface uses local system time at the time the value is being recorded.

History Recovery

The operation of the Batch Interface can be interrupted without loss of data. While the Interface is offline, the data is being buffered by the data sources such as Event Journal files.

The Interface can recover data, provided the data is still available in the data sources. If the interruption occurred while the interface was running, the data is recovered automatically

without user intervention. To perform historical data recovery, the Interface must be run in `Recovery` mode. In this mode, the Interface can recover data for any time period specified by you. The recovery mode is enabled by specifying the recovery time period through the command line parameters **`/rst=<date and time>`** (required) and **`/ret=<date and time>`** (optional). Data recovery is limited by BES historical data availability and a few other factors on the PI Server, including the number of licensed tags and the size and time frame of PI archives into which data is backfilled. Refer to Data Recovery section for more information.

SetDeviceStatus

The Health Point with the attribute ExDesc = [UI_DEVSTAT] tracks the status of the source devices. This tag is automatically created and configured by the interface if its missing on startup. The following events can be written into the tag:

- "Good" - the interface is properly communicating and reading data from the data sources.
- The following events represent proper communication with the data sources. This message is displayed on successful connection to each source.

  "2 | Connected/No Data | EVT Directory Monitor: <directory name> Initialized."
- The following list of events represents the failure to communicate with either the Event Journal file directory or Position directory, or failure to read data from the Event Journal File:

  "3 | 1 device(s) in error | Error monitoring directory (onError): <directory name>"

  "3 | 1 device(s) in error | Error monitoring directory: <directory name>"

  "3 | 1 device(s) in error | Failed to start directory monitoring thread: <directory name>"

  "3 | 1 device(s) in error | Error in scanning directory: <directory name>"

  "3 | 1 device(s) in error | Error obtaining EVT files EOF."

  "3 | 1 device(s) in error | Error getting current EVT file timestamp."

  "3 | 1 device(s) in error | Error reading EVT file: <filename>."

  "3 | 1 device(s) in error | Error while reading EVT file."

Vendor Software Required

The Batch Executive System (BES) and its accompanying support software are required for proper operation of this Batch interface.

Device Point Types

The interface receives data from the source as string data, and coerces the string data into numerical equivalents according to Tag Templates if defined.

Diagram of Hardware Connection

Figure 1. Schematic of Recommended Hardware and Software Configuration for Batch interface with Event Files as sources.

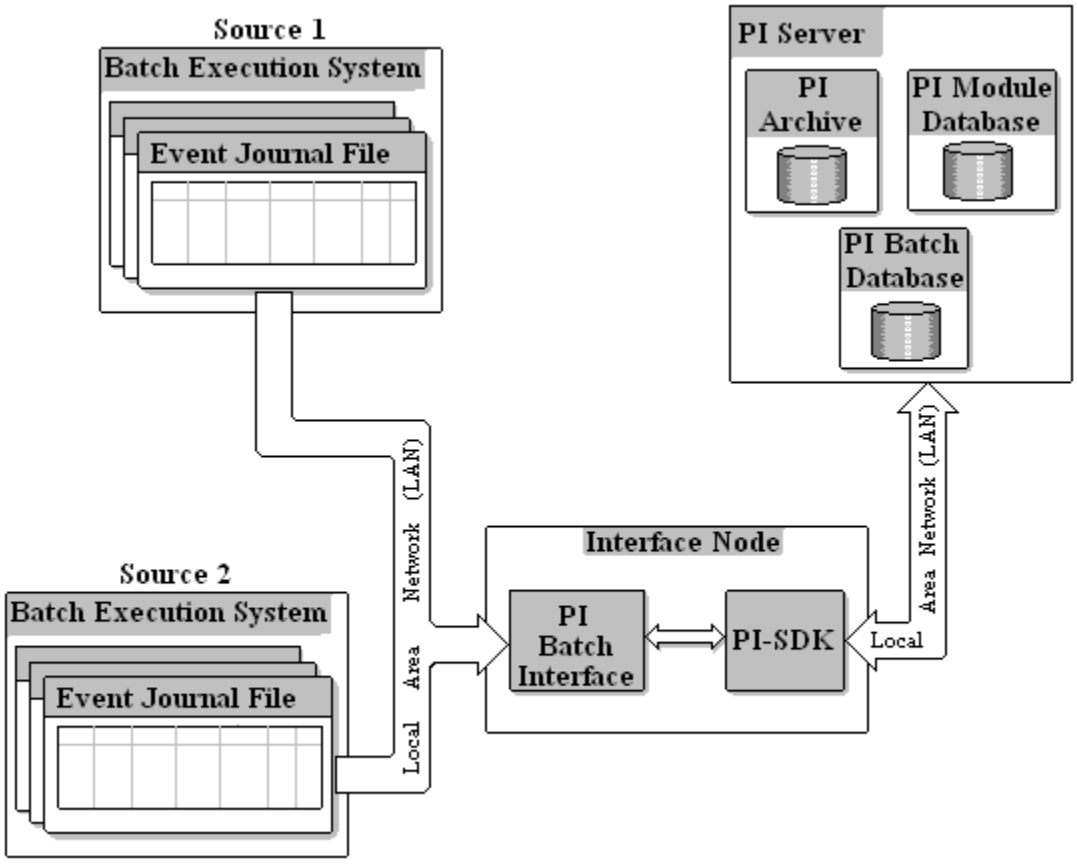
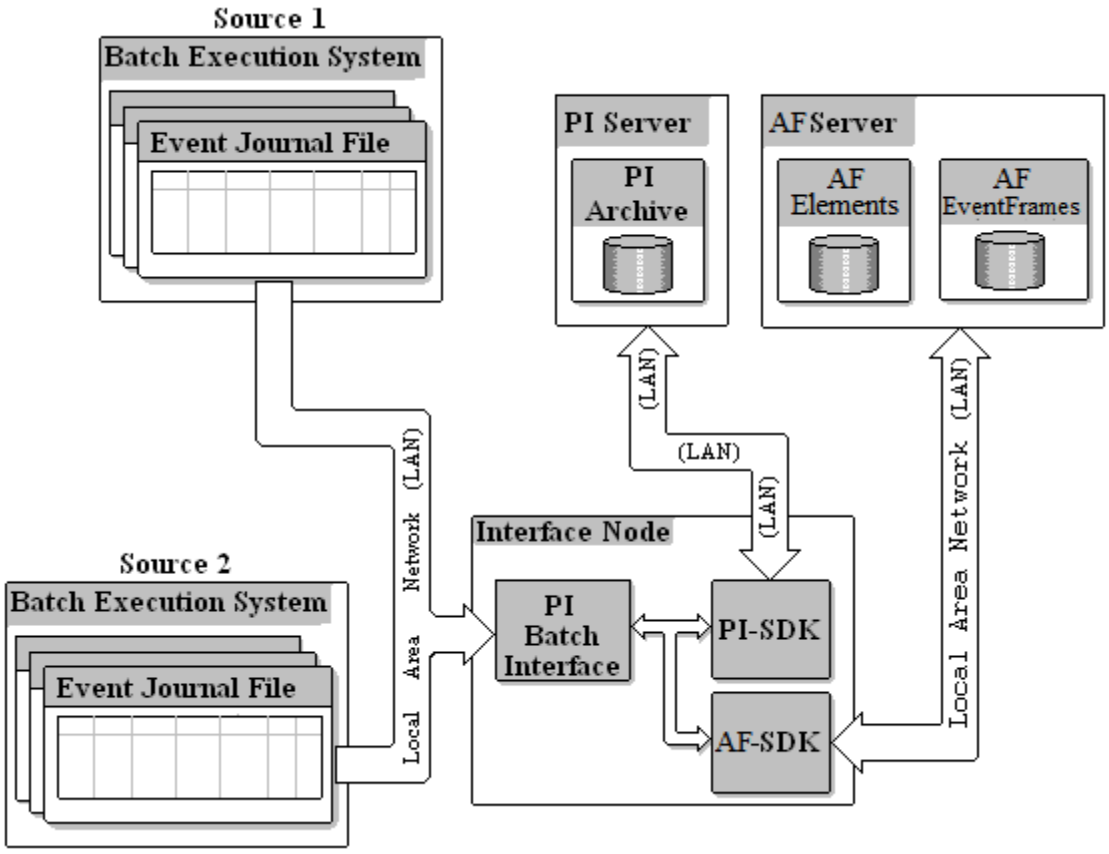


Figure 1b. Schematic of Recommended Hardware and Software Configuration for Batch interface with event files as data source and AF Server as host for asset and batch data.



The Batch interface can be installed on the same node as the batch execution system (BES) or on a completely separate node. To minimize contention for system resources, do not install the interface on a node where the PI Server is running. Contact the vendor of your BES for recommendations about installing third-party software such as the Batch Interface on the same node as the Rockwell FactoryTalk Batch Executive..

Chapter 2. Principles of Operation

This section describes the primary logic of the Batch interface.

Interface Modes

The Interface can be run in five different modes:

- RealTime (default)
- Recovery
- Preprocess
- Statistics
- Delete

RealTime mode is the default mode of operation and **Recovery** mode is designed to recover historical batch and tag data, provided the data still exists on the source. The principal difference between **RealTime** and **Recovery** modes is that, in **RealTime** mode, the interface synchronizes newly-acquired data from the source with the PI Server at the end of each scan, regardless of batch completion on the source. In **Recovery** mode, the interface synchronizes the batch only when it has completed on the source—that is, when the end time is known.

In **Recovery** mode, open batches are processed only when there are no completed batches left to be processed, and processing has reached the current time. The interface starts in **Recovery** mode. Recovery starts from the timestamp of the event last processed to the PI Server before shutdown and continues until the current time. If recover end time is omitted, the interface switches to **Realtime**. If an end time is specified, the interface recovers data for the specified time span and then exits.

Preprocess mode is designed for situations when source data must be written to PI archives using timestamps that are earlier than the primary PI archive. Due to the nature of the PI Server, newly-added tags, units and modules are indexed (referenced) only in the primary PI archive. Older archives do not include these modules, units and tags. In **Preprocess** mode, the interface creates modules, units, tags and tag aliases without processing batch data and adding events into the tags. After preprocessing, you must reprocess older archives with the offline archive utility. Please refer to the *PI Server System Management Guide* for details on archive reprocessing procedure (note that these procedures have changed with PI Server 2012). Preprocessing creates indexes for newly-added units, modules, tags in each reprocessed archive. You must run the interface in preprocess mode before writing new batch data to older PI archives. To run the interface in preprocess mode, specify the **`/mode=noupdate`** command line parameter in conjunction with the Recovery Start Time switch **`(/rst=<date and time>`**. To ensure all tags and modules are created, omit the Recovery End Time **`/ret=<date and time>`** parameter.

In **Statistics** mode, the interface compares source data with the PI server data. In this mode the interface does not write or modify any data on the PI Server. Upon completion, the

interface reports its results and exits. To run the interface in statistics mode, specify the start time, end time if desired, and the **/mode=stat)** command line parameter.

In **Delete** mode, the interface cleans PI archives based on specified source data only, leaving data from all other sources intact. Use delete mode only if the interface is unable to synchronize source batch data with the PI server. To run the interface in delete mode, specify start and end time and the **/mode=delete** command line parameter.

Multiple Data Sources

The Batch interface can process data coming from multiple sources simultaneously. This parallel processing is designed primarily for processing data from distributed-control Batch Execution Systems. For example, the control logic of a manufacturing process can be split between upstream and downstream segments, with each segment controlled by a separate FactoryTalk Batch Executive system. Even though the logical batch is the same, the actual batch-related data is split between two batch historians. This interface can merge data for such batches and store it in a single PI batch. Refer to Merging Multiple Source batches into a Single PIBatch for more details.

Parallel data processing also solves the problem of shared unit control, where overlapping batch recipes access same unit in different stages of their production cycles. This solution is achieved by acquiring data for the same time frame from multiple sources and combining time-ordered data using a single interface instance.

Data source(s) are configured in the INI file associated with the interface instance. The full path to the directory with EVT files is sufficient to configure a data source.

Table 1. Data source usage and description.

Object Name	Property name	Description
Source[#]		Defines the interface data source, where # is the index of the source. Must be a positive integer.
	.evtdir= [directory path] Required for EVT data Source	Defines the Event File journal directory associated with particular source. Example: Source[1].evtdir = D:\TEST\RELEASE\test_1 Source[2].evtdir = D:\TEST\RELEASE\test_2 Source[3].evtdir = D:\TEST\RELEASE\test_3

Event Journals as Data Source

Event journals are files that are generated directly by a FactoryTalk Batch Execution System (BES). Each file represents execution of particular recipe and contains a log of batch events as well as batch related data. The interface expects that each record (row) in the event file will contain at least 19 tab-delimited columns which contain the following information in the following order:

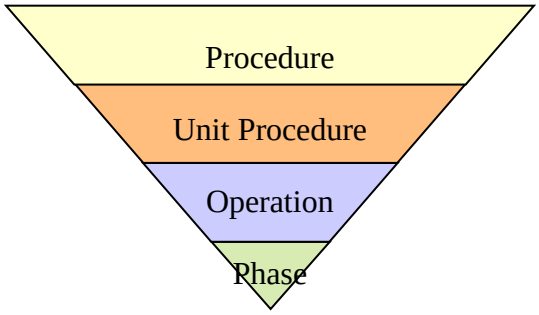
Column1: Timestamp (either LclTime or GMTTime)
Column2: BatchID
Column3: Recipe
Column4: Descript
Column5: Event
Column6: PValue

Column7: EU
Column8: Area
Column9: ProcCell
Column10: Unit
Column11: Phase
Column12: PhaseDesc
Column13: UserID
Column14: UniqueID
Column15: MaterialName
Column16: MaterialID
Column17: LotName
Column18: Label
Column19: Container

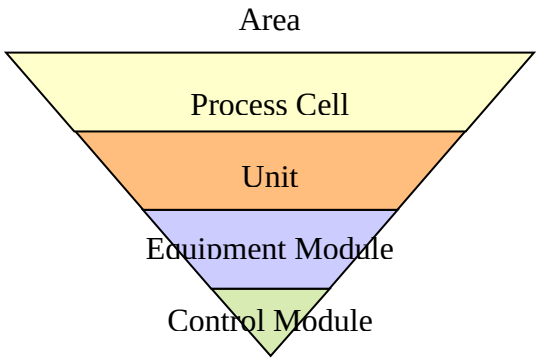
Recipe Model

The Recipe model, which describes batch processes, is a hierarchy of the procedures that are performed during the execution of a recipe, as shown in the following figure. In the S88 standard, the use of procedures and unit procedures is optional: a recipe can consist solely of operations and phases.

Figure 2. Recipe Model hierarchy



Equipment Model hierarchy



The Batch interface uses the S88 terminology and hierarchy as a framework to collate and store information in a structured manner in the PI Server (as Module and Batch databases) or the AF Server (as elements and event frames).

The interface maps a unit procedure to a PI UnitBatch. Only one unit procedure can be active in a unit at any time. This approach restricts the configuration of recipes that can be run by the BES, if the interface is to process the resultant data and populate the BDB in a meaningful

manner. If there are overlapping Unit Procedures on the same unit, the interface closes the conflicting PI UnitBatches, although the data is still processed into closed PI UnitBatches. The actual end time for truncated UnitBatch is stored in its Product property. The actual Product is appended by the keyword "_TrueEndUTC=", followed by the actual End Time for the specific unit batch specified as UTC seconds.

When the interface is configured to store batch data in the AF server, there are no restrictions on how the batch data can be stored, because AF event frames support parallel unit procedures natively.

If the recipe is divided into multiple smaller unit procedures or operations, enable merging for the interface. Please refer to the Merging Multiple Source batches into a single PIBatch section for more information on how the merge works.

PI Batch Database Methodology

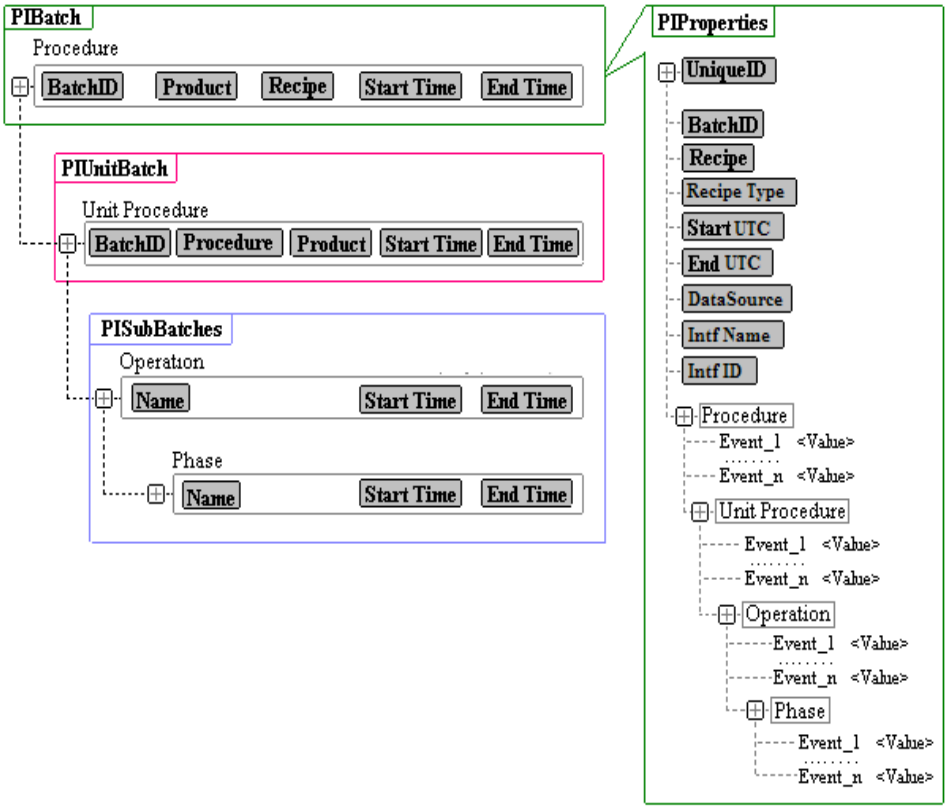
The PI Module and Batch Databases are used to organize and store batch data. Further discussion of these databases can be found in the PI 3.3 Data Archive Manual and the PI SDK tutorial documentation. To represent recipe procedures, unit procedures, operations, phases, phase states and phase steps, this interface creates PIBatch, PIUnitBatch and hierarchy of PISubBatch objects in the PI Batch Database (Fig. 4). Each of the objects created in the PI Batch Database has the following properties:

- Name (PISubBatch)
- Batch ID (PIBatch and PIUnitBatch objects only)
- Start time
- End time

In a PIBatch the name is stored in the Recipe property and, in a PIUnitBatch, the Procedure property stores the name of the corresponding recipe level. If illegal characters **(* ' ? | ` ")** are encountered in the BatchID, Name, Product, Recipe or Procedure fields, they are replaced with the underscore "_" character.

Each object in the PI Batch Database represents a specific level of the Recipe Model. However, the relationship between the PI Batch Database and the Recipe Model is complicated by the possibility of building a recipe without the procedure or unit procedure levels. In cases where the highest recipe level is an operation or phase (that is, neither the procedure nor unit procedure levels are defined), the interface still creates PIBatch and PIUnitBatch objects.

Figure 3. Schematic of PI Batch Database organization.



PIBatch

The PIBatch object is created for each batch defined in the data source. All records associated with the source batch can be recorded in the PIProperties collection of the PIBatch or in PI Points. The root PIProperty nodes are always the UniqueID of the batches which is assigned automatically by the Batch Executive. The interface stores the following batch properties under UniqueID: BatchID, Product, Formula Name, Recipe, Recipe Type, Start Time UTC, End Time UTC, Interface Name, Interface ID, DataSource, and events defined by the client. The underlying structure of the PIProperties collection is organized to reflect the hierarchy of the Recipe Model described by the data source where the Recipe names create hierarchical PIProperty nodes. Events of interest are stored in lists under appropriate Recipe node. Each PIProperty event name is defined as 'Event_<event count>', where <event count> is the current number of events already stored under specific node. This method of naming events is dictated by the PIProperty rule, which states that each event name under the same node must be unique. The PIProperty value can be defined through the use of Property templates. Please refer to Property Template section below for description and configuration steps.

The PIBatch represents the procedure within the recipe. Each PIBatch contains a collection of associated PI UnitBatches (which correspond to the Unit Procedures in the recipe).

The PIBatch object can represent a merged object, which contains multiple source batches with identical BatchIDs or a common subset of characters in the BatchID. The PI Batch Product and Recipe properties contain data associated with the first source batch that started the merged PI Batch. Use PIProperties to retrieve original source batch properties. For each

merged source batch, the interface creates a node named using the UniqueID of the source batch containing the original batch properties.

Note: Because the source batch can terminate unexpectedly without proper unloading by the operator, the interface maintains this batch in the local memory for 100 days. After 100 days, the batch is considered abandoned and the interface closes it with the latest known timestamp for this particular batch. The abandon timeout can be changed using the **`/abto=<days>`** (**A**bandoned **B**atch **T**ime **O**ut) command line parameter.

PI Batch Start Event Combinations

Data Source	PIBatch Start-Triggering Event(s)
FactoryTalk EVT	The batch recipe event containing: [Event] field = “System Message” and [Pvalue] field = “Beginning Of BATCH”. The associated [EU] field = “Procedure” / “Unit Procedure” / “Operation” / “Phase” determines the type of the particular recipe.

PI Batch End Event combinations

Data Source	PIBatch End-Triggering Event(s)
FactoryTalk EVT	The first out of two recipe events is used to set an End Time for PIBatch object. 1) The batch recipe event containing [Event] field = “System Message” and [PValue] field = “End Of BATCH” 2) The batch recipe event containing: [Event] field = “State Change” and [PValue] field = “REMOVED”/ ”COMPLETE” / ”ABORTED”

PIUnitBatch

A PIUnitBatch is created for each unit procedure defined in the data source. The start and end times of a PIUnitBatch are intended to reflect the start and completion of physical processing within a unit.

The PIUnitBatch properties do not change if the parent object is a merged PI Batch. A PIUnitBatch always contains the original BatchID and Procedure name as defined in the source, unless overridden using the **`/tbid`** command line parameter, which enforces a stripped BatchID to be used for PIUnitBatch objects and for all events to be stored in PIPoints and PIProperties.

When Operation- or Phase-level recipes are run, the interface uses the Operation/Phase name as the PIUnitBatch Procedure name.

PI UnitBatch Start Event Combinations

Data Source	PIBatch Start-Triggering Event(s)
FactoryTalk EVT	For Procedure, Unit Procedure level recipes, the following two events must be preset to set the Start Time for PIUnitBatch. The latest timestamp is used as the start time. 1) The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Unit Procedure Started”. 2) The arbitration event containing [Event] field = “Arbitration”, [Descript] field = “Unit Acquired”. The [PValue] field contains the actual unit name. For Operation level recipes the following two events must be present to start PIUnitBatch: 1) The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started”. 2) The arbitration event containing [Event] field = “Arbitration”, [Descript] field = “Unit Acquired” with the [PValue] field containing the actual unit name. For Phase level recipes, single event is used to set the Start Time for PIUnitBatch containing [Event] field = “State Change”, [PValue] field = “RUNNING”. The [Recipe] field contains the batch recipe hierarchy.

PI UnitBatch End Event combinations

Data Source	PIBatch End-Triggering Event(s)
FactoryTalk EVT	For Procedure, Unit Procedure level recipes, the first out of the following two events is used to set an End Time for PIUnitBatch: 1) The batch recipe event containing [Event] column = “System Message” and [Descript] column = “Unit Procedure Finished”. 2) The arbitration event containing [Event] field = “Arbitration”, [Descript] field = “Unit Released”. The [PValue] field contains the actual unit name. For Operation level recipes the first out of the following two events is used to set an End Time for PIUnitBatch: 1) The batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished”. 2) The arbitration event containing [Event] field = “Arbitration”, [Descript] field = “Unit Released” with the [PValue] field containing the actual unit name. For Phase level recipes, single event is used to set an End Time for the PIUnitBatch, containing [Event] field = “State Change” and [PValue] field = “COMPLETED” / “ABORTED” / “STOPPED”. 1) The [Recipe] field contains the batch recipe hierarchy.

PISubBatches

Operation

A PISubBatch is created for each source operation found within the data source as child for PIUnitBatch object.

Note: The operation and phase level recipes populate upper levels of PIBatch Database hierarchy automatically with PIUnitBatch Procedure property and PISubBatch operation name as the name of the source Operation/Phase recipe object.

PISubBatch Operation Start Event

Data Source	PISubBatch Operation Start-Triggering Event(s)
FactoryTalk EVT	For Procedure, Unit Procedure, Operation level recipes, the batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Started” is used to set the Start Time for PISubBatch operation level object. For Phase level recipes the batch recipe event containing [Event] field = “State Change” and [PValue] field = “RUNNING” is used to set the Start Time for PISubBatch operation level object. The [Recipe] field contains the batch recipe hierarchy.

PISubBatch Operation End Event

Data Source	PISubBatch Operation End triggering event(s)
FactoryTalk EVT	For Procedure, Unit Procedure, Operation level recipes, the first event out of two following events is used to set an End Time for PISubBatch operation level object: 1) the batch recipe event containing [Event] field = “System Message” and [Descript] field = “Operation Finished” 2) The batch recipe event containing [Event] field = “State Change” and [PValue] field = “REMOVED” (at Operation level). Note, this event is used due to possibility that some “Operation Finished” events are not present in EVT data source. For Phase level recipes the batch recipe event containing [Event] field = “State Change” and [PValue] field = “RUNNING” is used to set the Start Time for PISubBatch operation level object. The [Recipe] field contains the batch recipe hierarchy.

Phase

A PISubBatch is created for each phase found within the data source as child for Operation level PISubBatch object.

Note: The phase level recipes populate upper levels of PIBatch Database hierarchy automatically with PIUnitBatch Procedure property and PISubBatch operation name as the name of the source Phase recipe object.

PISubBatch Phase Start triggering events

Data Source	PISubBatch Phase Start triggering event(s)
FactoryTalk EVT	For Procedure, Unit Procedure, Operation, Phase level recipes, the batch recipe event containing [Event] field = “State Change” and [PValue] containing any value except: IDLE, READY, COMPLETE, ABORTED, REMOVED, STOPPED is used to set the Start Time for PISubBatch phase level object. The [Recipe] field contains the batch recipe hierarchy.

PISubBatch Phase End triggering events

Data Source	PISubBatch Phase End triggering event(s)
FactoryTalk EVT	For Procedure, Unit Procedure, Operation, Phase level recipes, the batch recipe event containing [Event] field = “State Change” and [PValue] field = “COMPLETE” or ”STOPPED” or ”ABORTED” or “REMOVED” is used to set an End Time for PISubBatch phase level object. The [Recipe] field contains the batch recipe hierarchy.

Phase State

A PISubBatch is created for each phase state found within the data source as child for Phase level PISubBatch object. All Phase States are sequential; start of new Phase State ends the previous Phase State. Note, the self-terminating Phase States which set its End Times are COMPLETE, ABORTED, STOPPED and REMOVED.

PISubBatch Phase State triggering events

Data Source	PISubBatch Phase State triggering event
FactoryTalk EVT	The batch recipe event containing [Event] field = “State Change” and [PValue] field = <State Name>. The [Recipe] field contains the batch recipe hierarchy.

Phase Step

A PISubBatch is created for each phase step found within the data source as a child for the Phase State level PISubBatch object. Phase Steps are not S88 complaint and are custom to each particular implementation and configuration of the Batch Execution System. By default this level of PISubBatches is not enabled. To enable this feature use the optional switch **`/ras=<Start Substring>, <End Substring>`** (**R**eport **A**s **S**tep). The Phase Steps are always created beneath the first PISubBatch Phase State = "RUNNING", regardless if the parent Phase State is ended or not. The Phase Step name and start/stop events are coming from the "Descript" column. The triggering event is "Report". The Phase Steps do not create the higher level PI Batches, UnitBatches and SubBatches, if the parent Phase is not found. If the Phase Step was not closed by the appropriate closing event, it will be closed by the end of the parent Operation level PI SubBatch. 0-duration Phase Steps are ignored. Multiple sequential Start/End events are ignored except the first one.

PISubBatch Phase Step Start triggering events

Data Source	PISubBatch Phase State Start triggering event
FactoryTalk EVT	The following two events can set the Start Time for PISubBatch phase step object. 1) The event containing [Event] field = "Report" and [Descript] field containing **<Start Substring>**. The Phase Step name is determined as the prefix substring to <Start Substring> in [Descript] field. 2) The event containing [Event] field = "Report" and [PValue] field containing **<Start Substring>**. The Phase Step name is determined as the prefix substring to <Start Substring> in [PValue] field. The [Recipe] field contains the batch recipe hierarchy.

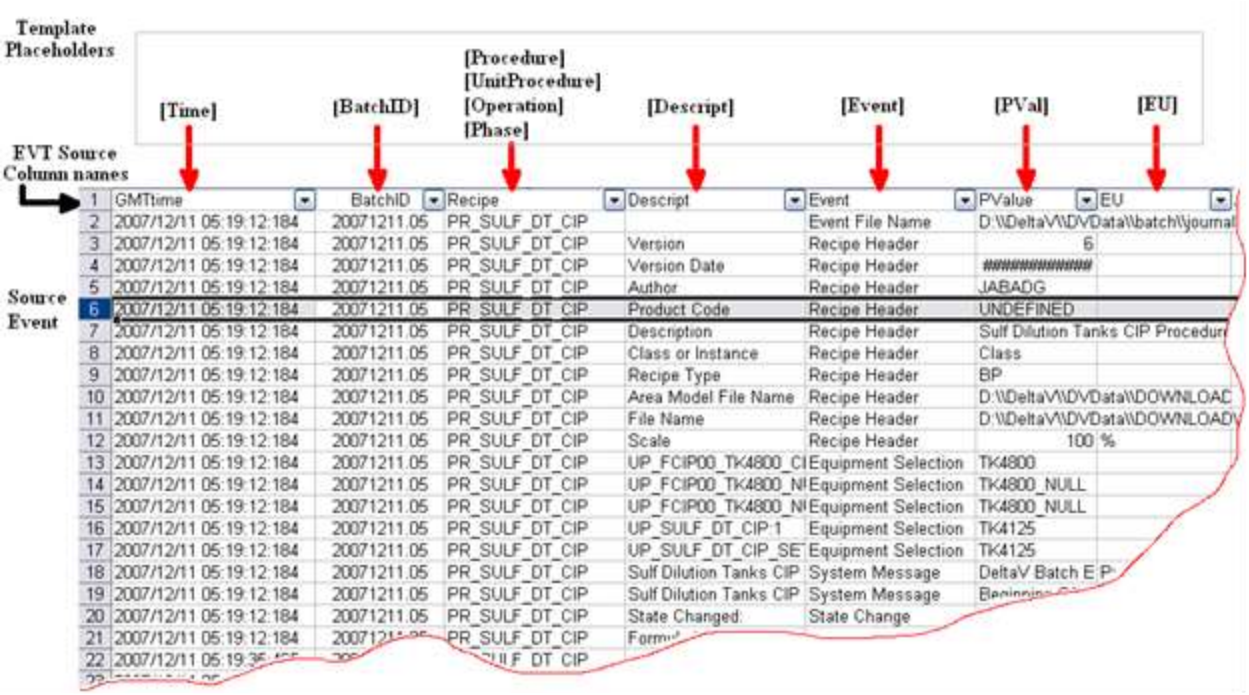
PISubBatch Phase Step End triggering events

Data Source	PISubBatch Phase State Start triggering event
FactoryTalk EVT	The following two events can set an End Time for PISubBatch phase step object. 1) The event containing [Event] field = "Report" and [Descript] field containing **<End Substring>**. The Phase Step name is determined as the prefix substring to <End Substring> in [Descript] field. 2) The event containing [Event] field = "Report" and [PValue] field containing **<End Substring>**. The Phase Step name is determined as the prefix substring to <End Substring> in [PValue] field. The [Recipe] field contains the batch recipe hierarchy.

Template Placeholders

The Batch interface uses templates to specify what is stored in PI Batch Properties or PI AF Attributes, and PI Points. Templates can also define the equipment hierarchy structure in the PI Module Database or PI AF. A template defines the custom name and/or value structure applied to particular PI object. The template is defined using a combination of a free text and placeholders. The placeholder can be referred as to the name of the column in EVT data source, except the Recipe column which is broken down into subcolumns, such as Procedure, UnitProcedure, Operation and Phase.

Figure 4. Example of Placeholders and associated EVT columns



	GMTime	BatchID	Recipe	Descript	Event	PValue	EU
2	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP		Event File Name	D:\\DeltaV\\DVData\\batch\\journal	
3	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version	Recipe Header	6	
4	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Version Date	Recipe Header	##########	
5	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Author	Recipe Header	JABADG	
6	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Product Code	Recipe Header	UNDEFINED	
7	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Description	Recipe Header	Sulf Dilution Tanks CIP Procedur	
8	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Class or Instance	Recipe Header	Class	
9	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Recipe Type	Recipe Header	BP	
10	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Area Model File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOAD	
11	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	File Name	Recipe Header	D:\\DeltaV\\DVData\\DOWNLOADV	
12	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Scale	Recipe Header	100	%
13	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_CI	Equipment Selection	TK4800	
14	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
15	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_FCIP00_TK4800_NI	Equipment Selection	TK4800_NULL	
16	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP:1	Equipment Selection	TK4125	
17	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	UP_SULF_DT_CIP_SE	Equipment Selection	TK4125	
18	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	DeltaV Batch E P	
19	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Sulf Dilution Tanks CIP	System Message	Beginning C	
20	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	State Changed:	State Change		
21	2007/12/11 05:19:12:184	20071211.05	PR_SULF_DT_CIP	Formul			
22	2007/12/11 05:19:36		ULF_DT_CIP				

Example template:

```
Sample [Time] | [Descript]-[BatchID]:[Event]__pvalue:[PVal][EU]
```

This structure contains the free text and the placeholder combination. Assume that the incoming event is the row number 6 (Figure 8), which is **Recipe Header**. Then using the template structure we can replace placeholders with the actual data from the associated columns to create the following text:

```
Sample 2007/12/11 05:19:12:184 | Product Code:Recipe
Header__pvalue:UNDEFINED
```

Note, in this example [EU] placeholder was replaced with BLANK value since the source row did not have the associated column populated.

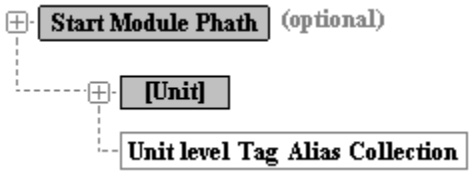
PIBatch and PIUnitBatch Product Property

The ProductID information coming from Event Journal files is stored as the PValue in the row that contains the description "Product Code". This event is typically a Recipe Header event.

PIModule Creation

The interface automatically creates modules and units as required. PI Units (PIModules with the IsUnit flag set to true) are created if needed when they are first encountered in the data source. The interface maintains only the unit modules. By default, the placement of these modules is at the root level of the Module DB. You can define the root Starting Module Path using the **`/smp`** command line parameter. The following figure shows the default structure of the PI module hierarchy created by the interface.

Figure 5. Interface PI Module DB Structure



The Batch Interface references PI tags at the unit and phase PIModules through tag aliases if the tag name contains the unit module name or unit module name mask.

If this default equipment hierarchy is not feasible, you can specify a custom hierarchy using the Recipe[2].ModulePath template, which is defined in the INI file associated with a specific interface instance.

Note: To override the default module path, specify
`Recipe[2].ModulePath=<custom equipment path>`.

PI AF Event Frames Methodology

In this approach, PI Event Frames organize and store the batch data. The event frame hierarchy supports the storage of source batch data with no data manipulation. At any level of the hierarchy, each event frame has its own set of attributes, which provide the ability to store source batch attributes under a specific event frame regardless of its depth. This approach is a major improvement over the PI Batch database, where only the top level object (PIBatch) can contain attributes (PIProperties).

Each event frame has the following fields:

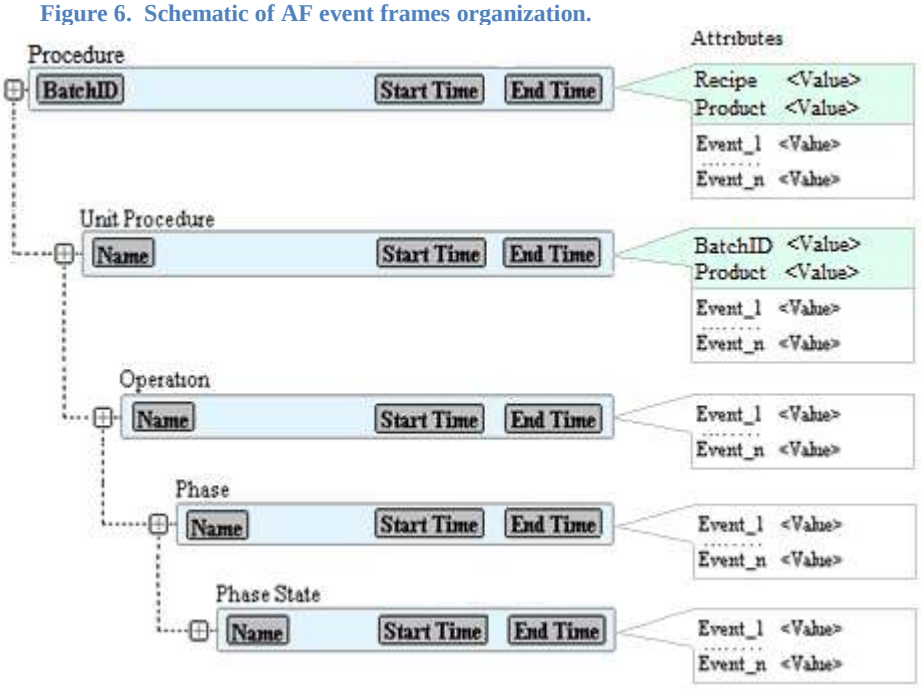
- Name
- Description
- Start Time
- End Time
- Template
- Category

- Event Specific Attributes
- Referenced elements (such as Unit, Phase Module)

In the root-level event frame, the Name field contains the source batch BatchID. For lower-level event frames, the Name field contains the actual recipe name, such as UnitProcedure, Operation, Phase, etc. To maintain compatibility with the way data is stored in PI Batch Database, the other source batch properties are stored as event frame attributes. For Procedure-level recipe, source batch Product and Recipe properties are stored as event frame attributes. For UnitProcedure-level recipes, the source batch BatchID and Product are stored as event frame attributes.

Any illegal characters **(* ‘ ? | ` “)** in the Name field are replaced with underscores. Each object in the event frame represents a specific level of the Recipe Model. However, the relationship between the event frames and the Recipe Model is complicated by the possibility of representing a recipe that lacks procedure or unit procedure levels. If the highest recipe level is an operation or phase (that is, neither procedure nor unit procedure levels are defined), event frames that correspond to Procedure and UnitProcedure level must be still created by the interface.

Figure 6. Schematic of AF event frames organization.



Procedure

The root event frame is created for each batch defined in the data source, and represents the Procedure in the Recipe. Each root event frame contains a collection of associated child event frames that correspond to the Unit Procedures in the recipe. All records associated with the source batch can be recorded in the Attributes collection of the event frame or in PI Points. Because source batches can have identical BatchID and Recipe Names within the same timeframe, the interface stores additional information in the Extended Properties of the root event frame to match the source batch with an event frame. The Extended Properties are a flat Name-Value collection. The Name is the BatchID of the source batch, which is automatically assigned by the Batch Executive, and the value is the XML structure containing the following batch properties: BatchID, Product, Formula Name, Recipe, Recipe Type, Start Time UTC, End Time UTC, Interface Name, Interface ID, DataSource.

To maintain compatibility with PI Batch database, the root event frame Name is the BatchID of the source batch. The "Recipe" (Procedure Name) and "Product" properties are stored as the searchable attributes of the event frame. Below is the source batch property mapping to an AF event frame.

Source Procedure Properties	Event Frame Fields	Event Frame Attributes	Referenced Elements
BatchID	Name		
Procedure Name		Recipe	
Product		Product	
Start Time	Start Time		
End Time	End Time		
	Template="Procedure"	default Attributes: Recipe, Product	
	Category="OSIBatch"		

In addition to "Product" and "Recipe" Attributes, by default, the interface captures the following batch-associated events and stores them in Procedure-level AF event frame Attributes:

AF Attribute Name	Value mapped to source column	Category
Version	[MBRVERSION]	Information
VersionID	[MBRVERSIONID]	Information

The Procedure-level event frame can represent a merged object, which contains multiple source batches with identical BatchIDs or a common subset of characters in its BatchID. The Product and Recipe attributes contain data associated with the first source batch that started the merged event frame. For each merged source batch, the interface creates a node in Extended Properties of the event frame, named with the UniqueID of the source batch and the value containing the XML containing the original source batch properties.

Note: A source batch can terminate unexpectedly without proper unloading by the operator. The interface maintains such a batch in the local memory for 100 days, after which the batch is considered abandoned and the interface closes the batch with the latest known time stamp for this particular batch. The abandon timeout can be changed using the command line parameter **`/abto=<days>`** (**A**bandoned **B**atch **T**ime **O**ut).

UnitProcedure

A UnitProcedure-level event frame is created for each unit procedure as defined in the data source. Each UnitProcedure-level event frame is created as a child of the Procedure-level event frame and contains the subset of event frames that represent the source batch Operation-level recipe. The start and end times of an event frame are intended to reflect the onset and completion of physical processing in a unit. The parallel UnitProcedures are supported completely by AF event frames. That is, the event frame End Time reflects the actual end time of the source UnitProcedure,

The name field of the UnitProcedure-level event frames reflects an actual source batch UnitProcedure name. To maintain compatibility with PI Batch database, the interface stores the "BatchID" and the "Product" source batch properties as searchable attributes of the event frame. The following table shows how a source unit procedure is mapped to event frame fields and attributes.

Source UnitProcedure Properties	AF Event Frame Fields	AF Event Frame Attributes	Referenced Elements
BatchID		BatchID	
UnitProcedure Name	Name	Procedure	
Product		Product	
Start Time	Start Time		
End Time	End Time		
Unit			Unit
	Template="UnitProcedure"	default Attributes: BatchID, Procedure, Product	
	Category="OSIBatch"		

The UnitProcedure-level event frame properties do not change if the parent object is a merged event frame. UnitProcedure event frames always contains the original BatchID and Procedure name as defined in the source, unless the **`/tbid`** parameter was specified in the command line. This parameter configures a stripped BatchID to be used for UnitProcedure event frames objects and for all events to be stored in PIPoints and event frame attributes.

When Operation or Phase-level recipes are run, the interface uses the Operation/Phase name as the UnitProcedure-level event frame name.

Operation Level

An Operation-level event frame is created for each Operation as defined in the data source. Each Operation-level event frame is created as a child of the UnitProcedure-level event frame, and contains the subset of event frames that represent the source batch Phase-level recipe.

The Name field of these event frames reflects the source recipe Basic Function name. The following table shows how a source operation recipe is mapped to event frame fields and attributes.

Source Operation Properties	Event Frame Fields	Event Frame Attributes	Referenced elements
Operation Name	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
	Template name depends on the level in hierarchy. For Operation level Template=”Operation”, for Phase: Template=”Phase”, etc.		
	Category=”OSIBatch”		

Phase

A Phase-level event frame is created for each Phase, as defined in the data source. Each Phase-level event frame is created as a child of the Operation-level event frame and contains the subset of event frames that represent the source batch Phase States-level recipe.

The name field of the Phase-level event frame reflects an actual source recipe Phase name. Below is the source phase recipe mapping to event frame fields and attributes:

Source Phase Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced elements
Phase Name	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
Phase Module			Phase Module
	Template=”Phase”		
	Category=”OSIBatch”		

Phase State

A Phase State-level event frame is created for each Phase State, as defined in the data source. Each Phase State-level event frame is created as a child of the Phase-level event frame and, if configured, can contain the subset of event frames that represent the Phase Step objects. All Phase States are sequential; the start of new Phase State ends the previous Phase State.

The self-terminating Phase States that set its end times are COMPLETE, ABORTED and STOPPED. These Phase States have a zero-duration timeframe.

The name field of the Phase State event frames reflects an actual source recipe Phase State name. Below is the mapping of source phase state to event frame fields and attributes:

Source Phase State Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced Elements
Phase State	Name		
Start Time	Start Time		
End Time	End Time		
Unit			Unit
Phase Module			Phase Module
	Template=”Phase State”		
	Category=”OSIBatch”		

Phase Step

An event frame is created for each Phase Step found in the data source as a child of the Phase State-level event frame. Phase Steps are not S88-compliant and are unique to each particular implementation and configuration of the Batch Execution System. By default, this level of event frames is not enabled. To enable this feature, specify the optional switch **`/ras=<Start Substring>, <End Substring>`** (**R**eport **A**s **S**tep). Phase Steps are always created beneath the first Phase State EventFrame Name = “RUNNING”, regardless of whether the parent Phase State is ended. The Phase Step name and start/stop events come from the “Descript” column. The triggering event is “Report”. If the parent Phase is not found, Phase Steps do not create the higher-level Procedure-, UnitProcedure-, Operation- or Phase-level event frames. If the Phase Step was not closed by the appropriate closing event, it is closed by the end of the parent Operation-level event frame. Zero-duration Phase Steps are ignored. Multiple sequential Start/End events are ignored, except the first one.

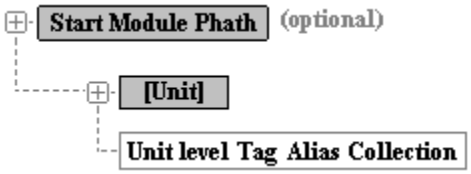
Below is the source phase step mapped to event frame fields and attributes:

Phase Step Properties	AF EventFrame Fields	AF EventFrame Attributes	Referenced elements
Unit			Unit
Phase Module			Phase Module
	Template=”Phase Step”		
	Category=”OSIBatch”		

PI AF Element Creation

The interface creates the PI AF elements that represent the source equipment hierarchy. PI AF elements are created if needed when they are first encountered in the data source. The following elements in the equipment hierarchy are maintained by the interface: Area, Process Cell, Unit and Phase Module. By default, these elements reside at the root level of the element collection. You can define an optional starting element path under which the equipment hierarchy is created by specifying the **/smp** command line parameter. The default structure of the PI AF elements hierarchy used by the interface is depicted in the following figure.

Interface PI AF Element Structure



The interface automatically references PI tags for the unit and phase modules PI AF elements through tag aliases if tag names contain unit and phase module names. If the tag name refers to the unit and not the phase module, only the Unit Alias for this tag is created.

If the default equipment hierarchy is not feasible for any reason you can create a custom equipment hierarchy using an equipment template, which is specified in the INI file associated with the interface instance.

Foreign Language Support

The Batch interface supports languages other than English using a look-up table for parameters and values. No translation is required to populate the PI Batch and PI Module database.

The language translation syntax is as follows:

```
translate: <English value> = <translation>
```

Using translations, the interface can create and populate PI Tags and PI Properties from templates defined using the native language. For example, consider the following tag template:

```
Tag[1].Name = [UnitID] abc_[Parameter,value="Bericht"]
Tag[1].Value = [Value]:def
Tag[1].type = string
Tag[1].unitalias = Some Bericht
Tag[1].Descriptor = Bericht for Unit: [UnitID]
Tag[1].EngUnits = just text

Property[1].Value = [TimeStamp] [Parameter,value="Bericht"]
[UnitID]-[Value]
```

These templates are triggered by "Bericht" (the German word for "Report"), which is found in the data source, and the tag name and PI Property value are based on the native language parameter name. For example, for a row with a unit field containing "U101" and an

associated value field containing “testing”, the resulting PI Tag name is “U101 abc_Bericht” and the resulting tag value is “testing:def”.

With the use of language translations we can create tag where foreign word/phrase is replaced with translated word/phase. For example:

```
translate:  "Bericht" = "Report"
translate:  "testing"  = "1"
```

The resulting tag name is "U101 abc_Report" and the resulting tag value is "1:def".

The same logic applies to property template definitions. Translations are not case-sensitive. Language translations don’t have to be defined before tag or property templates; they can be defined anywhere in the INI file.

Initialization File

The initialization (INI) file is where you specify the interface configuration, including data sources, translations, and templates for products, equipment, tags and properties. The INI file is named `PIFTBInt<serviceid>.ini`. Specify each parameter on its own line, using the syntax *`parameter = setting`*. Precede comments with two forward slashes (//).

Following is an example INI file. For details about specific settings, refer to the descriptions of templates and batch data processing in this chapter.

```
[General]

[Source Template]

source[1].evtdir = “C:\test\evt”
source[1].evtdir = \\testbox2\journals\evt


// [Basic Tag template, triggered on Event=Report, aliases are
created as tag name]
Tag[1].Name  = [Unit]_[PhaseModule]_Report
Tag[1].Value = [Pval]
Tag[1].Type  = float


// [Tag template with custom aliases, triggered on Event=Owner
Change]
Tag[2].Name  = [Unit]_[PhaseModule]_Owner Change

Tag[2].Value = [time]_[Descript]

Tag[2].Type  = string

Tag[2].unitalias = [PhaseModule] Owner Change Me

Tag[2].phasealias = Owner Change Me

// [Tag template with custom aliases, triggered on set of
events defined as triggers]

// [Note: Unitalias and Phasealias are NOT going to be created
since there are no Unit or Phase
// Module defined in the tag name]
Tag[3].Name = Generic Tag
```

```
Tag[3].Value = [time]_[Event]_[BatchID]_[pval]
Tag[3].Type = string
Tag[3].trigger = Report
Tag[3].trigger = Owner Change
Tag[3].trigger = Operator Prompt
Tag[3].unitalias = [phasemodule] abcd
Tag[3].phasealias = testing
[Property Template]
Property[1].Value = [Time] State Change  [Descript] [pval]
```

Event Logging

The interface can store incoming events to the PI Properties hierarchy of the PI Batch or AF Event Frame Attributes when the events match triggering events defined in Property Templates. The interface can create new PI Tags (and link them to a unit and phase module with a PI Aliases) when incoming events match triggering events defined in Tag Templates.

In the following sections, "Placeholder" indicates the data source column name that is available to interface. The placeholder is normally delimited by square brackets [ ]. Angle brackets (< >) indicate an exact match in any field of an incoming event. All placeholders are replaced by the actual field data during processing.

The following tables describe supported placeholders.

Generic Placeholders

Place Holder	Description
[TIMESTAMP]	Timestamp of the event. For internal interface events [EVENT, value="PIEVENT"] this placeholder refers either to start or end of the destination time interval. For all other events this placeholder contains the timestamp of the event..
[TAG]	Refers to PI Server PI Point.

Data Source - Parameter Data Placeholders

Placeholder	Description
[DESCRIPT]	Description
[EVENT]	Event
[PVAL]	Parameter value
[EU]	Engineering units
[AREA]	Process area
[PROCESSCELL]	Process cell
[UNIT]	Unit name
[PHASEMODULE]	Phase name
[USERID] or [USER]	User name
[UNIQUEID]	UniqueID

Placeholder	Description
[MATERIALNAME]	
[MATERIALID]	
[LOTNAME]	
[LABEL]	
[CONTAINER]	

Interface Internal Placeholders

These placeholders are applicable when the triggering expression contains `[Parameter, value="PIEVENT"]`.

Place Holder	Description
[BATCHID]	String value that is stored as PIBatch BatchID and PIUnitBatch BatchID property. For a top-level Event Frame, it is the Name property. For second-level Event Frame, it refers to the Attribute "BatchID".
[PROCEDURE]	Refers to value stored at level=1 of the batch hierarchy. For PIBatch DB this is PIBatch "Recipe" property. For AF Database, this is top level Event Frame "Recipe" Attribute.
[UNITPROCEDURE]	Refers to value stored at level=2 of the batch hierarchy. For PIBatch DB this is PIUnitBatch "Procedure" property. For AF Database, this is Event Frame "Name" property.
[OPERATION]	Refers to value stored at level=3 of the batch hierarchy. For PIBatch DB this is PISubBatch "Name" property. For AF Database, this is Event Frame "Name" property.
[PHASE]	Refers to value stored at level=4 of the batch hierarchy. For PIBatch DB this is PISubBatch "Name" property. For AF Database, this is Event Frame "Name" property.
[PHASESTATE]	Refers to value stored at level=5 of the batch hierarchy. For PIBatch DB this is PISubBatch "Name" property. For AF Database, this is Event Frame "Name" property.
[PHASESTEP]	Refers to value stored at level=6 of the batch hierarchy. For PIBatch DB this is PISubBatch "Name" property. For AF Database, this is Event Frame "Name" property.
[UNIT]	Refers to the name of the unit. For Module DB this is PIModule "Name" property. For AF Database, this is AF Element "Name" property.

The following wildcards can be used in any property field of Tag or Property Templates.

Wildcard	Description
#	single digit numerical value (0-9)
@	single alpha character (a-z, A-Z)
?	any single valid symbol
*	An array of valid symbols
!	repeat previous mask symbol

For example:

```
Tag[1].Name = [Parameter, value="Temp_Sens*"
```

Note that specifying `[Parameter]` in a trigger expression is equivalent to specifying `[Parameter, value="*"]`. For maximum efficiency, specify an exact match. For example:

```
Tag[1].Trigger = [Parameter, value="Recipe Data"]
```

Advanced Parsing Parameters

Each placeholder can contain parameters that parse incoming data. The syntax is as follows:

`[placeholder, <comma-separated parameter list>]`

The names of parameters, placeholders, and value substrings are not case-sensitive. If additional parameters are used for at least one placeholder, then in case of resulting substring returning empty set, the whole template will be set to blank. To search all the fields of an incoming event, specify the * wildcard for the placeholder.

The following table lists valid parameters.

Parameter	Description
VALUE="substring" or "mask"	Defines the value to search for in a particular column. Masks are allowed. If '*' (search all event fields) is used instead of Name of Placeholder [*,value="test"] is equivalent to <test>
LBE="substring" Optional	**L**eft **B**ound **E**xclusive substring. Defines the left bound of the target substring value. The resulting substring does not include the LBE defined boundary substring.
LBI="substring" Optional	**L**eft **B**ound **I**nclusive substring. Defines the left bound of the target substring value. The resulting substring includes the LBI defined boundary substring.
RBE="substring" Optional	Right Bound Exclusive substring. Defines the right bound of the target substring value. The resulting substring does not include the RBE defined boundary substring.
RBI="substring" Optional	Right Bound Inclusive substring. Defines the right bound of the target substring value. The resulting substring includes the RBI defined boundary substring.
Delim="substring" Optional	Delimiter character or substring. Must be used in conjunction with the Count parameter. This parameter defines the field separator. If used, it narrows the resulting substring to the substring contained within delimiters, where the starting delimiter index is specified by the count parameter. **Note**: Right and left boundary substrings can be specified as well, to parse the delimited substring.
Count=# Optional	Index of the delimiter from which to start parsing. Must be used in conjunction with the Delim parameter.

For example, assume that [Value] column field contains the following data:

```
|U:browntod|C:SP_CHARGE_AMOUNT|O:1200|N:1123|E:kg|M:Local
```

The following table shows examples of placeholder parameter combinations and the resulting data.

Placeholder syntax	Resulting substring
[value]	|U:browntod|C:SP_CHARGE_AMOUNT|O:1200|N:1123|E:kg|M:Local
[value, lbe="N:"]	1123|E:kg|M:Local
[value, lbi="N:"]	N:1123|E:kg|M:Local
[value, rbe="tod"]	|U:brown
[value, rbi="tod"]	|U:browntod
[value, lbe="U:", rbe="|"]	Browntod
[value, lbi="U:", rbe="|"]	U:browntod
[value, lbe="O:", rbi="kg"]	1200|N:1123|E:kg
[value, delim="|",count=3]	O:1200
[value, delim="|",count=3,lbe="O:"]	1200
[value, delim="|",count=2,lbe="C:SP",rbe="UNT"]	_CHARGE_AMO
[value, delim="|",count=6,lbe="M:"]	Local

Attribute/Property Templates

When the interface is configured to use a PI AF Server, batch data is stored as event frames and batch-associated data is stored in AF attributes. AF attributes are part of each event frame that enable batch data to be stored with the each event. All levels of the event frame hierarchy can store batch data in AF attributes.

When the interface is configured to run only against the PI Server, batch-recipe-associated data can be stored at the PIBatch level (root level) of the recipe hierarchy only by using the PIProperties collection, due to PI Server limitations. To maintain the recipe hierarchy, PIProperties are organized as a recipe tree, where each PIProperty node is the name of the recipe level, (procedure, unit procedure, operation, or phase). The data is stored in name-value lists under each node.

Note: The batch PI Properties collection has a limitation of 1Mb per PIBatch object. To avoid this limitation, do not store all incoming events into a batch PIProperties collection.

By default the interface does not store batch associated data in PIProperties. To store data in PIProperties, use Property Templates that define the subset of events and the associated PIProperty value structure for each event to be stored in PIProperties. The Property Templates are not case sensitive and are defined in the INI file associated with the interface instance. The Property Template can define only PIProperty values, not PIProperty names. Each PIProperty event name under the same PIProperty node must be unique. Event names are assigned as "Event_<event count>", where <event count> is the current number of events already stored under the PI Property node.

To configure the templates, use the `Attribute[index]` or `Property[index]` keywords as follows:

```
Property[index].Name = free text with or without placeholders
(hierarchy supported) (optional)
Property[index].Value = free text with or without placeholders
Property[index].Trigger = free text with or without placeholders
Property[index].Translate = true/false (default: false)
Property[index].EngUnits = free text with or without placeholders
(AF only)
Property[index].Type = integer/float/string/auto
Property[index].Category = free text with or without placeholders
(AF only)

Attribute[index].Name = free text with or without placeholders
(hierarchy supported) (optional)
Attribute[index].Value = free text with or without placeholders
Attribute[index].Trigger = free text with or without placeholders
Attribute[index].Translate = true/false (default: false)
Attribute[index].EngUnits = free text with or without
placeholders (AF only)
Attribute[index].Type = integer/float/string/auto
Attribute[index].Category = free text with or without
placeholders (AF only)
```

Specify placeholder names in square brackets. The triggering expression must be embedded in the Value Structure or specified through explicit Trigger(s). Specifying multiple placeholders in a single triggering expression is treated as AND logic and specifying multiple trigger expressions is treated as OR logic.

If the source Engineering Units (UOM) do not match the AF Server Units of Measure (UOM), define a conversion using the UOMMAP keyword. The syntax is as follows:

```
UOMMAP: <Source UOM> = <AF UOM>
```

Example:

```
UOMMAP: \\B0\\C = DEGC
```

To write Properties under the UniqueID PIProperty node, regardless of the source recipe sublevel from which the triggering event originated, specify "$" as the first element in name path, as shown below:

```
Property[1].Name = $\[Parameter]
Attribute[1].Name = $\[Parameter]
```

To write Properties under the PIBatch root PIProperties, regardless of the level of the source recipe from which the triggering event originated, specify the @ symbol as the first element in name path, as shown below:

```
Property[1].Name = @\[Parameter]
Attribute[1].Name = @\[Parameter]
```

Property Template Description

Template Name	Allowed Placeholders	Description
`Property[#].Name` **Or** **`Attribute[#].Name`** Optional	[TIME] [BATCHID] [PROCEDURE] [UNITPROCEDURE] [OPERATION] [PHASE] [PHASESTATE] [PHASESTEP] [DESCRIPT] [EVENT] [PVAL] [EU] [AREA] [PROCESSCELL] [UNIT] [PHASEMODULE] [USERID] or [USER] [UNIQUEID] [MATERIALNAME] [MATERIALID] [LOTNAME] [LABEL] [CONTAINER] [*,value="Exact Field"], [*,value="Field Mask"], or advanced parsing	Defines the Name structure of the PIProperty. The triggering expression or Event Type must be embedded in the value structure. PIProperty Names under the same PIProperty must be unique. If Template Property - Name is not defined, the PI Property names are created automatically by the interface as **`Event_(Same Node Event Count)`**. If Name is defined and there is an event that results in a PIProperty Name already existing in PI Server, the interface replaces the existing PIProperty value with the new one. Each incoming event can trigger multiple Property Templates, if defined in each template as a triggering event. In the Name property, the hierarchy of names is supported. **Example** **`Property[1].Name = Materials\[Parameter]`** If the Property Template is triggered, the interface creates under proper Recipe PIProperty – PI Property "Materials" and as child property – the value of the [Parameter] placeholder. By default all properties are placed under the particular recipe nodes in PIProperties that correspond to the recipe structure created in PIBatch database. Starting with version 2.0.0.1, parameter data from any level of the source recipe hierarchy can be placed at the root level of the PIBatch PIProperties object by specifying the $ as the first node name in path. For example: **`Property[1].Name = $\[Parameter]`** If [Parameter]="Recipe Data" and the Property Template is triggered, the interface creates the property named as "Recipe Data" under the specific root UniqueID PIProperty
`Property[#].Value` **Or** **`Attribute[#].Value`** Required	Same as for Name. And `[TAG]`	Defines the value structure of the PI Property. The triggering expression or Parameter Name must be embedded in the value structure. Because PI Property Names under the same PI Property Node must be unique, the property names are created automatically by the interface. Each incoming event can trigger multiple Property Templates, if defined in each template as a triggering event. Property Template Value is a string with optional placeholders. The placeholder is the name of the

Template Name	Allowed Placeholders	Description

		source column. For each incoming event, the placeholder is replaced by the corresponding field from the event structure.
		You can specify the exact value or a value mask using wildcards. If no match is found in predefined/any event fields, the whole property template is ignored.
		For the "State Change" parameter, the Property Template can be defined as follows:
		`Property[1].Value = [BatchID] | event: [Parameter, value="State Change"] | val: [Value]`
		Or using a mask:
		`Property[1].Value = [BatchID] | event: [Parameter, value="State Ch*"] | val: [Value]`

`Property[#].Trigger`
Or
`Attribute[#].Trigger`
Optional
	Same as for Name Property except `[TIME]`	Defines the triggering expression or Parameter Name that used to create and populate PI Properties. If trigger is defined, it overrides any triggering expression in Value property. You can define multiple triggers for a single template property.
		To trigger a particular template, the interface uses the placeholders embedded in the expression, which are treated as AND logic. Use multiple triggering expressions to create OR logic.
		Example:
		`Property[1].Trigger = [Parameter, value="State Change"]`
		`Property[1].Trigger = [Value, value="test"]`
		Using mask:
		`Property[1].Trigger = [Parameter, value="State Ch*"]`
		`Property[1].Trigger = [Value, value="tes*"]`

`Property[#].Translate`
Or
`Attribute[#].Translate`
 	
Optional
	Values:
	`true/false`
		To enable translation, set to true.

`Property[#].Type`
Or
`Attribute[#].Type`
 	
Optional
	`String`
	`Float`
	`Integer`
		Defines the type of the PIProperty or AF Attribute depending on whether the PI Server or AF Server hosts batch data.

`Property[#].Category`
Or
`Attribute[#].Category`
	Same as for Name property	Defines the AF Attribute Category property. This property can be used for grouping of the attributes.

Template Name	Allowed Placeholders	Description
Optional AF Only		

`Attribute[#].UOM`
Or
`Attribute [#].EngUnits`
Or
`Attribute[#].EU`
 	
Optional
 	
AF Only
	Same as for Name property	Defines the Engineering Units (Units of Measure) for the specific AF Attribute. Allowed placeholders are not case sensitive.

For example, assume the Property Template is defined in in the INI file as follows:

```
Property[1].Value=[TimeStamp]: Parameter:<REAC_TEMP*> |
V:[Value]_Testing
```

The index (1) identifies the template that was used to create this particular PIProperty event structure. The text string after the equal sign (=) specifies the structure.

Following is an incoming event from the data source:

```
[TimeStamp]=”12/01/2008 12:01:05”
[Parameter]=REAC_TEMP_SP
[Value]=25.00000
```

After the interface processes the event using the example template, the following PIProperty value is added to the PIBatch object:

```
12/01/2008 12:01:05: Parameter:REAC_TEMP_SP | V:25.0000_Testing
```

Tag Templates

The Batch interface can store batch-associated data in PI Points, commonly known as tags. Every Event Type emitted by the data source can be recorded in the PI Server. By default, interface does not create tags or populate them with events. You enable this functionality by defining Tag Templates in the INI file associated with each interface instance. The INI file must have the same filename and an extension of INI. By default the INI file is located in the same directory as the startup file. If it is in a different directory, specify its location using the **`/inifile=<full path filename>`** command line parameter.

Using Tag Templates you can define structures for tag name, tag data type, tag value, unit alias name, phase module alias name, engineering units and descriptor properties. The timestamp for each tag event is obtained directly from the data source. The tag name structure, tag value structure and tag type properties must be configured; all other properties are optional. If only tag name is defined, define the triggering “parameter name” as part of the tag name structure. If an explicit trigger is defined, the tag creation and population is based on the parameter type defined in the **.Trigger** property, overriding the parameter name in tag name (if defined). Multiple tag templates can be triggered by the same source “parameter name” and a single template can be triggered by multiple source “parameter names”.

Multiple tag templates can write to the same PI tag (if the **.Name** attribute of the tag templates resolves to the same PI tag name), which is useful when you want different values to be written to the same PI tag depending on the trigger for each.

Note: Explicit triggers override the Tag Name embedded triggering.

You can specify the tag value type. Valid types are float, integer and string. If the value type is not specified, the batch interface creates a string PI Point and treats all event values as strings.

```
Tag[index].<Property> = Free text
```

The index also serves as the Location2 value in the PI Point attributes and identifies which Tag Template created the point.

Possible Tag Template <Property> definitions:

```
Tag[index].Name = Name structure (with embedded triggering Event
Type or Event Type Mask or Expression)
Tag[index].Value = Event value structure as free text
Tag[index].Trigger = Event Type or Event Type mask or Expression
Tag[index].Type   = string/integer/float
Tag[index].UnitAlias = unit tag alias name structure (default: as
.Name)
Tag[index].Descriptor = value structure as free text (default:
blank)
Tag[index].EngUnits = value structure as free text (default:
blank)
Tag[index].Translate = true/false   (default: false)
Tag[index].Annotation = free text with or without placeholders
Tag[index].Annotation2 = free text with or without placeholders
```

If the name structure contains placeholders, the tag template is triggered only if the incoming event contains values for all the placeholders in the name structure.. The event value structure does not have this limitation: placeholders can be replaced with empty fields unless you have configured advanced field value parsing.

The following table lists the properties, values and placeholders that can be used to define value/name structures.

Tag Template Description

Template Property Name	Allowed Placeholders	Description

`Tag[#].Name`
 	
Required
	[BATCHID]
	[PROCEDURE]
	[UNITPROCEDURE]
	[OPERATION]
	[PHASE]
	[PHASESTATE]
	[PHASESTEP]
	[DESCRIPT]
	[EVENT]
	[PVAL]
	[EU]
	[AREA]
	[PROCESSCELL]
	[UNIT]
	[PHASEMODULE]
	[USERID] or [USER]
	[UNIQUEID]
	[MATERIALNAME]
	[MATERIALID]
	[LOTNAME]
	[LABEL]
	[CONTAINER]
	[*,value="Exact Field"],
	[*,value="Field Mask"],
	advanced parsing
		Defines the name structure of the tag. The triggering Parameter or expression can be specified in either the Tag[#].Name or in Tag[#].Trigger properties.
		The tag name structure can contain the exact word or phrase (specified within angled brackets <…> or as [*,value="…"]) to be found in any fields of the incoming event. If the column is known, use the advanced parsing described above. For example, desired column is "Event" and value is "Report", the placeholder can be defined as [Event,value="Report"]. The word or phrase can be also detected by specifying a mask with wildcards.
		For example, assume the incoming Descript column contains a field called B10_OP_CIP100. To create a tag when this descriptor is encountered, specify the tag name template as follows:
		`Tag[1].Name = [unitid] <B10_OP_CIP100> REPORT_RATE.`
		Or, using a mask:
		`Tag[1].Name = [unitid] <B10_OP_CI*> REPORT_RATE.`
		The triggering event can be specified using mask; for example:
		`Tag[1].Name = [unitid] <B10_OP_CI*> <REPORT_R*>`
		Each incoming event can be used to create/populate multiple PI Tags, if it is defined as a triggering event in multiple Tag Templates.

`Tag[#].Value`
 	
Required
	Same as Name, and
	`[TIME]`
	`[TAG]`
		Defines the event value structure for the specific PI Point. The event timestamp is taken from the incoming event's [TimeStamp] field.

`Tag[#].Type`
 	
Required
	`String`
	`Float`
	`Integer`
		Defines the type of the PI Point to be created and how to treat the events written to this tag. For Compliance Suite tag template, this property must be set to "string"

Template Property Name	Allowed Placeholders	Description
Tag[#].Trigger Optional	Same as for Name property except [TIME]	Defines the triggering parameter name or expression used to create and populate PI tags. If a trigger is defined, it overrides any triggering parameter or expression in the Name property. There can be multiple triggers defined for a single template tag. Placeholders in expressions trigger particular templates. Placeholders are treated as AND logic. To specify OR logic, use multiple triggering expressions.. Example: Tag[1].Trigger = State Change Using a mask: Tag[1].Trigger = State Ch* Using a triggering expression with two placeholders: Tag[1].Trigger=[Event,value="State*] [Pval,value=RUNNING"] This expression triggers the tag template only if both conditions are met.
Tag[#].UnitAlias Optional	Same as for Name property	Defines the unit-level alias name structure for a specific template tag. The field can be specified as an exact phrase or using a mask. Starting with interface version 1.0.1.0, an optional sub unit module path can be specified in the alias name. Use backslashes to separate parent and child modules and use the pipe ("|") symbol to separate the module path and the actual alias name. By default, the interface uses the Name property as the unit-level alias name and the [unitid] module as the alias location. The names for PI Aliases must be unique. Starting with version 1.0.2.0, you can create aliases on PI modules based on an absolute module path: specify the '$' sign as the first module in the module path. '$' stands for root module, which can be specified using the /smp=<Start Module Path> command line parameter. If the /smp is omitted, '$' defaults to the PI MDB root node. **Example 1**: This alias is going to be created on particular [Unitid] module with alias name as State alias Tag[1].UnitAlias = State alias **Example 2**: This alias is going to be created under [Unitid]\ABC\def with alias name template as State alias Tag[2].UnitAlias = ABC\def | State alias **Example 3:** If no module root is specified and [Unitid]="U101, the interface is going to create hierarchy as (PI MDB) \ abc_U101 And place an alias under "abc_U101" node. Tag[3].UnitAlias = $ \ abc_[Unitid] | State alias
Tag[#].Descriptor Optional	Same as for Name property	Defines the Tag Descriptor structure for the specific PI Point.
Tag[#].EngUnits Optional	Same as for Name property	Defines the Engineering Units (EngUnits) structure for the specific PI Point.

Template Property Name	Allowed Placeholders	Description
`Tag[#].Translate` Optional	Values: true/false	Set to true to enable translation of words and phrases in Name, Value, UnitAlias, PhaseAlias, Descriptor and EngUnits.
`Tag[#].Annotation` Optional	Same as Name property	Enables you to annotate each value written to PI Server using the specific Tag Template. The annotation is written to PI as a string value. Example: `Tag[1].Annotation = [BatchID]`
`Tag[#].Annotation 2` Optional	Same as Name property	Enables you to annotate each value written to PI Server using the specific Tag Template. The annotation is written to PI as a NameValue object. Example: `Tag[1].Annotation2 = [BatchID]`

Example:

The Tag Template is defined in INI file as follows:

```
Tag[1].Name= Test Set Point [Unitid]
[Parameter,value=”REAC_TEMP*”]
Tag[1].Value= P: [Parameter] | V:[Value] | Testing
Tag[1].Type = string
Tag[1].UnitAlias = Temperature Set point for [Parameter]
Tag[1].EngUnits = oC
Tag[1].Descriptor = Sample Temperature Set Point for
Reactor:[Unitid]
Tag[1].Annotation=[BatchID]
```

Assume that incoming event from data source contains the following data:

```
[BatchID]=Batch_123
[TimeStamp]=”12/01/2008 12:01:05.123”
[Parameter]=REAC_TEMP_SP
[Value]=25.00000
[Unitid]=U101
```

The resulting PI Tag name is “Test Set Point U101 REAC_TEMP_SP”

Because the [Unitid] placeholder is defined in Tag Name Template, the interface finds or adds the following alias for PI Tag on unit U101: “Temperature Set point for REAC_TEMP_SP”.

When the PI Tag and alias are verified, the following event value is added to the point:

Event Timestamp	Event Value
12/01/2008 12:01:05.123	P: REAC_TEMP_SP | V:25.00000 | Testing

Because the Annotation property is define in the tag template, the preceding value is annotated with the text “Batch_123”

Example

In the following scenario, the parameter name must not be in the tag name and the tag type must be float. The following template configures the desired behavior:

```
Tag[1].Name= Test Set Point [Unitid]
Tag[1].Value= [Value]
```

```
Tag[1].Type = float
Tag[1].Trigger = [Parameter,value=”REAC_TEMP*”]
Tag[1].UnitAlias = Temperature Set point for [Parameter]
Tag[1].EngUnits = oC
Tag[1].Descriptor = Sample Temperature Set Point for
Reactor:[Unitid]
```

The interfaces uses the preceding template to process the following incoming event:

```
[TimeStamp]=”12/01/2008 12:01:05.123”
[Parameter]=REAC_TEMP_SP
[Value]=25.00000
[Unitid]=U101
```

The Tag template is triggered when the parameter = “REAC_TEMP_SP” is found in the incoming event. When the template is triggered, the actual PI Tag name is set to “Test Set Point U101”. Because the `[Unitid]` placeholder is defined, the interface finds or adds the following alias for the PI Tag on unit U101: “Temperature Set point for REAC_TEMP_SP”.

When the PI Tag and alias are verified, the following event value is added to the point:

Event TimeStamp	Event Value
12/01/2008 12:01:05.123	25.0

Tag Templates – Logging PI Batch Database Activity

The Batch Interface can log its activity in the PI Batch database by generating its own PIEvents. These events are based on the logic that the interface uses to trigger PI Batches, PIUnitBatches, and PISubBatches (Operations, Phases). You can configure Tag Templates triggered by these PIEvents to write batch-triggering data to PI tags, which can be used for reporting purposes in PI client tools.

PIEvent records have the following placeholders and values, which can be used in the **.Trigger** attribute of the tag template:

Placeholder	Values	Description
[EVENT]	PIEVENT	All PIEvents must trigger on [EVENT, value="PIEVENT"]

[DESCRIPT]	BATCH
	UNITBATCH
	OPERATION
	PHASE
		The DESCRIPT column contains the batch level you want to trigger on. For example:
		`[DESCRIPT, value="UNITBATCH"]`
		Or
		`[DESCRIPT, value="PHASE"]`

[PVAL]
Or
[VALUE]
	START
	END
		The PVAL column contains either the start event or end event associated with the defined DESCRIPT. For example:
		`[PVAL, value="START"]`
		Or
		`[PVAL, value="END"]`

Multiple tag templates can write to the same PI tag, if the .Name attribute of the tag templates resolves to the same PI tag name. This feature is useful when you want different values to be written to the same PI tag dependent on the trigger for each. For example, a value of 1 can be

written to the tag when the UnitBatch starts and a value of 0 can be written to the same tag when the UnitBatch ends.

The following placeholders are useful when defining the tag template (especially for the .Value tag template attribute):

Placeholder	Description
[BATCHID]	The Batch ID Name
[PRODUCT]	The Product Name
[PROCEDURE]	The PIBatch Procedure (Recipe) Name
[UNITPROCEDURE]	The PIUnitBatch Procedure Name
[OPERATION]	The Operation Name
[PHASE]	The Phase Name
[PHASESTATE]	The Phase State Name
[PHASESTEP]	The Phase Step Name

PIEVENT Example 1: PIBatch Active Tag

```
Tag[11].Name=BESName:PIEvent.Batch.Active
Tag[11].Value=BATCH START: [BatchID] |Prod: [PRODUCT] |Rec:
[PROCEDURE]
Tag[11].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="BATCH"] [PVAL,value="START"]
//// SAME TAG
Tag[12].Name=BESName:PIEvent.Batch.Active
Tag[12].Value=BATCH END: [BATCHID] |Prod: [PRODUCT] |Rec:
[PROCEDURE]
Tag[12].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="BATCH"] [PVAL,value="END"]
```

PIEVENT Example 2: PIUnitBatch Active Tag

```
Tag[21].Name=BESName:[UNIT].PIEvent.UnitBatch.Active
Tag[21].Value=1
Tag[21].Type=integer
Tag[21].UnitAlias=PIEvent.UnitBatch.Active
Tag[21].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="UNITBATCH"] [PVAL,value="START"]
//// SAME TAG
Tag[22].Name=BESName:[UNIT].PIEvent.UnitBatch.Active
Tag[22].Value=0
Tag[22].Type=integer
Tag[22].UnitAlias=PIEvent.UnitBatch.Active
Tag[22].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="UNITBATCH"] [PVAL,value="END"]
```

PIEVENT Example 3: PIUnitBatch BatchID Tag

```
Tag[31].Name=BESName:[UNIT].PIEvent.UnitBatch.BatchID
Tag[31].Value=[BATCHID]
Tag[31].UnitAlias=PIEvent.UnitBatch.BatchID
Tag[31].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="UNITBATCH"] [PVAL,value="START"]
```

```
//// SAME TAG
Tag[32].Name=BESName:[UNIT].PIEvent.UnitBatch.BatchID
Tag[32].Value=Inactive
Tag[32].UnitAlias=PIEvent.UnitBatch.BatchID
Tag[32].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="UNITBATCH"] [PVAL,value="END"]
```

PIEVENT Example 4: Phase Active Tag

```
Tag[41].Name=BESName:[UNIT].PIEvent.Phase.Active
Tag[41].Value=PHASE START:
[PROCEDURE]\[UNITPROCEDURE]\[OPERATION]\[PHASE]
Tag[41].UnitAlias=PIEvent.Phase.Active
Tag[41].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="PHASE"] [PVAL,value="START"]
//// SAME TAG
Tag[42].Name=BESName:[UNIT].PIEvent.Phase.Active
Tag[42].Value=PHASE
END:   [PROCEDURE]\[UNITPROCEDURE]\[OPERATION]\[PHASE]
Tag[42].UnitAlias=PIEvent.Phase.Active
Tag[42].Trigger=[EVENT,value="PIEVENT"] [DESCRIPT,
value="PHASE"] [PVAL,value="END"]
```

Using PI Tags as Data Sources

You can configure existing PI Tags as input data sources. Based on the batch event-triggering mechanism, the interface can read data from PI Tags and write results into new data structures defined by Tag and Property Templates. To configure a tag as a data source, use the following syntax:

[Tag, Name="PI Tag Name", <*list of parameters delimited by comma***>**

Valid parameters are as follows:

Parameter	Description
Name="string" Required	The name of the PI Tag from which data is retrieved.
Range="substring" Optional	The time frame for which the data is queried. It can be number of events, time frame or "PIOBJECT". "PIOBJECT" instructs the interface to use the time frame of the related PI batch/unitbatch/subbatch object Examples: Range="10": The last ten events from triggered batch event timestamp Range="10d": The events for last 10 days. Range="PIOBJECT": The events for the time frame of the related batch object start and end times are retrieved.
Func="substring" **Optional**	In conjunction with Range parameter, specifies the aggregation function to be used on retrieved data. Possible values for this parameter: "MIN": Minimum value over the range. "MAX": Maximum value over the range. "TOTAL": Sum of values over the range. "MID": Average of values over the range.

Advanced parsing parameters can be used in the [Tag] placeholder.

Property Template Example:

```
Property[1].Name = TestTagCalc
Property[1].Value = total:[Tag, name="sinusoid", range="10d",
func="TOTAL"] and min:[Tag, name="test_data_1", range="10d",
func="MIN"]
Property[1].Trigger = [Parameter, value="PIEVENT"] [Descript,
value="BATCH"] [Value, value="START"]
```

In this example, the Property Template is triggered on the internal event that is thrown when the PI Batch is created (started). This template creates a PI Property named "TestTagCalc" under the batch with string values from two tags: "sinusoid" and "test_data_1". If, for the specified time range, the sum of event values for "sinusoid" is 1000 and the minimum for "test_data_1" is -25.123, the following name and value combination is written to PI Batch Properties:

```
TestTagCalc = total:1000 and min:-25
```

Tag Template Example 1:

```
Tag[1].Name = Global Tester 1
Tag[1].Value = [Tag,name="test4_data", range="10d", func="total"]
Tag[1].Trigger = [Parameter, value="PIEVENT"] [Descript,
value="BATCH"] [Value, value="START"]
```

In this example, the Tag Template is triggered on the internal event that is thrown when the PI Batch is created (started). The result is written to a PI Tag named "Global Tester 1". If, for the time range, the sum of event values for the "test4_data" PI Tag is 1234, the following value is written to "Global Tester 1":

```
Timestamp: (batch start)
Value:     1234
```

Tag Template Example 2:

```
Tag[2].Name = Global Tester 2
Tag[2].Value = [Tag,name="test2_data", range="PIOBJECT",
func="total"]
Tag[2].Trigger = [Event,value="PIEVENT"] [Descript,
value="BATCH"] [Value, value="START"]
```

The result is calculated for the batch's full time range.

Tag Template Example 3:

```
Tag[3].Name = Global Tester 3
Tag[3].Value = [Tag,name="test2_data]
Tag[3].Trigger = [Parameter, value="State Change"] [Descript,
value="running"]
```

The Tag Template is triggered on the Siemens batch event "State Change" when the descriptor field is "RUNNING". The resulting tag name is "Global Tester 3" and the value is taken from PI Tag "test2_data" at the timestamp of the Siemens batch event.

Recipe Templates

Starting with version 2.0.0.1, the interface supports recipe templates. Recipe templates enable you to redefine the recipe name convention used for PIBatch, PIUnitBatch and PISubbatch object definitions. The syntax for recipe templates is as follows:

```
Recipe[index].Name= Free text
Recipe[index].BatchID = Free text
Recipe[index].ModulePath = Free text defining path
Recipe[index].Product = Free text
Recipe[index].ProductTrigger = Triggering expression
Recipe[index].Translate= true/false or 1/0
Recipe[index].Merge = true/false or 1/0
Recipe[index].Category = Free text
Recipe[index].Category[Index2].Name = Free text
Recipe[index].Category[index2].Trigger = Free text

Recipe[index].Template = Free text
Recipe[index].Template[Index2].Name = Free text
Recipe[index].Template[index2].Trigger = Free text

Recipe[index].Attribute[index2] – Enables you to define an
attribute template for specific Recipe level.
```

The index specifies the level in the recipe hierarchy, starting at 1 for procedure.

The following table lists the placeholders that can be used in a Name template.

Template Name	Allowed Placeholders in Value	Value Description
`Recipe[#].Name` Required	[BATCHID] [PROCEDURE] [UNITPROCEDURE] [OPERATION] [PHASE] [PHASESTATE] [PHASESTEP] [DESCRIPT] [EVENT] [PVAL] [EU] [AREA] [PROCESSCELL] [UNIT] [PHASEMODULE] [USERID] or [USER] [UNIQUEID] [MATERIALNAME] [MATERIALID] [LOTNAME] [LABEL] [CONTAINER] [*,value="Exact Field"], [*,value="Field Mask"], advanced parsing	Defines the naming convention used by the interface to create PIBatch, PIUnitBatch and PISubbatch objects. The index (#) specifies the level in the recipe hierarchy as follows: 1: Procedure (PIBatch Recipe field) 2: Unit Procedure (PIUnitBatch Procedure) 3: Operation (PISubBatch Name field) 4: Phase (PISubBatch Name field) Defaults: Recipe[1].Name=[Procedure] Recipe[2].Name = [UnitProcedure] Recipe[3].Name=[Operation] Recipe[4].Name=[Phase] **Example** **`Recipe[1].Name = abc_[Procedure]`** If the incoming event's [Procedure] field contains the value "Test", the PIBatch Recipe field is set to "abc_Test".
`Recipe[#].BatchID` Optional	Same as Name	Specifies the BatchID of the Recipe object. Supports the PIBatch BatchID and PIUnitBatch BatchID fields.

`Recipe[#].Descriptor`
Optional
`AF Only`
	Same as Name	Specifies the AF event frame Descriptor property for the source Recipe object.

`Recipe[#].ModulePath` Optional	Same as Name	Specifies the Module path of the Recipe object. Supports PIUnitBatch (level 2) only. The end module in the path is always treated as the Unit and marked as a PIUnit in the PI Module Database.
`Recipe[#].Product` Optional	Same as Name	Specifies the Product of the Recipe object. Supports the PIBatch and PIUnitBatch Product field. If ProductTrigger is not defined, this template is populated based on the data in the event that creates the Recipe object.

Template Name	Allowed Placeholders in Value	Value Description

`Recipe[#].ProductTrigger`
Optional
	Same as Name	Populates the Product field of the particular Recipe object after the object is created. Useful when the Product is defined by a separate event. Example:
		`Recipe[1].Product = [Value]`
		`Recipe[1].ProductTrigger = [Parameter, Value="Recipe Header"] [Descript, value="Product Name"]`

`Recipe[#].Translate` Optional	True (1) False (0)	Enable/disable translation of Names. By default, translation is disabled.
`Recipe[#].Merge` Optional	True (1) False (0)	Enables/disables merging of same-named objects under the same parent. Disabled by default.

`Recipe[#].Category`
Optional
 	
`AF Only`
	Same as Name	For each recipe level, defines the AF Event Frame Category. If the event that creates an event frame contains insufficient information, no category is assigned. To assign Category to an event frame after its creation, use the dynamic Category[x] property

`Recipe[#].Category[x].Name`
Optional
 	
`AF Only`
	Same as Name	For each recipe level, this dynamic property enables you to define the AF event frame Category based on any event that is related to particular recipe item. This property can create as many categories as desired. Index (x) is a positive integer that binds Name and Trigger(s) subproperties for a specific Category[x] property. If AF Category does not exist on the AF Server, the interface creates it. This property must be used with Recipe[#].Category[x].Trigger
		
		Example:
		`Recipe[1].Category[10].Name = SCR`
		`Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]`

`Recipe[#].Category[x].Trigger`
Optional
 	
`AF Only`
	Same as Name	Defines the triggering expression for a specific recipe category. There can be multiple triggers for a single Recipe[#].Category[x].Name. This property must be used with Recipe[#].Category[x].Name
		
		Example:
		`Recipe[1].Category[10].Name = SCR`
		`Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]`
		`Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_01"]`
		`Recipe[1].Category[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_02"]`

Template Name	Allowed Placeholders in Value	Value Description

`Recipe[#].Template`
Optional
 	
`AF Only`
	Same as Name	For each recipe level, this property enables you to define the AF event frame template. If sufficient information is not available in the event that creates an event frame, no AF template is assigned. To assign a template to an event frame after its creation, use the dynamic Template[x] property

`Recipe[#].Template[x].Name`
 	
Optional
 	
`AF Only`
	Same as Name	For each recipe level, this dynamic property enables you to define the AF event frame template based on any event that is related to particular recipe item. This property can assign only one AF template to a particular AF event frame. The interface uses the first matching Recipe[#].Template[x] property to be assigned to an event frame. This property must be used with Recipe[#].Template[x].Trigger.
		The index (x) is a positive integer that binds the Name and Trigger subproperties for the Template[x] property.
		
		Example:
		`Recipe[1].Template[10].Name = BATCH_A`
		`Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]`

`Recipe[#].Template[x].Trigger`
 	
Optional
 	
`AF Only`
	Same as Name	Defines the triggering expression for the AF event frame template. You can define multiple triggers for a single Recipe[#].Template[x]. This property must be used with Recipe[#].Template[x].Name
		
		Example:
		`Recipe[1].Template[10].Name = BATCH_A`
		`Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051"]`
		`Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_01"]`
		`Recipe[1].Template[10].Trigger = [Descript, value="Formula Name"] [Pval, value="SCR 20051_02"]`

Default (precompiled) Recipe Templates:

```
// batch
Recipe[1].Name = [Procedure]
Recipe[1].BatchID = [BatchID]
Recipe[1].Product = [Pval]
Recipe[1].ProductTrigger = [Event,value="Recipe
Header"][descript,value="Product Code"]
Recipe[1].Category = OSIBatch
```

```
Recipe[1].Template = Procedure

// unitbatch
Recipe[2].Name = [UnitProcedure]
Recipe[2].BatchID = [BatchID]
Recipe[2].Product = [Pval]
Recipe[1].ProductTrigger = [Event,value="Recipe
Header"][descript,value="Product Code"]
Recipe[2].ModulePath = [Area]\[ProcessCell]\[Unit]
Recipe[2].Category = OSIBatch
Recipe[2].Template = UnitProcedure

// subbatches
Recipe[3].Name = [Operation]
Recipe[3].Category = OSIBatch
Recipe[3].Template = Operation

Recipe[4].Name = [Phase]
Recipe[4].Category = OSIBatch
Recipe[4].ModulePath = [Area]\[ProcessCell]\[Unit]\[PhaseModule]
Recipe[4].Template = Phase

Recipe[5].Name = [PhaseState]
Recipe[5].Category = OSIBatch
Recipe[5].Template = PhaseState

Recipe[6].Name = [PhaseStep]
Recipe[6].Category = OSIBatch
Recipe[6].Template = PhaseStep
```

Recipe Template Example 1:

In this example, Recipe Templates redefines the PIBatch Recipe to be the concatenation of the Procedure and UniqueID fields from the data source. The PISubbatch Operation Name is redefined as the concatenation of the UnitProcedure and Operation fields from the data source.

```
Recipe[1].Name = [Procedure]_[UniqueID]
Recipe[3].Name = [UnitProcedure]_[Operation]
```

Recipe Template Example 2:

In this example, the Recipe Template defines the static PI AF Category "CAT_UNITBATCH" for the UnitProcedure level. If it is missing, this category is created and assigned to all UnitProcedure-level event frames regardless of the recipe type.

```
Recipe[2].Category = CAT_UNITBATCH
```

Recipe Template Example 3:

```
Recipe[1].Category[1].Name    = PROC_A
Recipe[1].Category[1].Trigger = [Procedure, value="DVProc:1-*"]
Recipe[1].Category[2].Name    = [Pval]
Recipe[1].Category[2].Trigger = [Descript, value="Product Code"]
```

In this example, the Recipe Template defines the dynamic PI AF Category for the root-level AF event frame based on the value of Procedure or Product Code from the data source. If the AF Category is not found on the AF Server, the interface creates it.

Recipe Template Example 4:

```
Recipe[2].Template = OSI_UnitProcedure
```

In this example, the Recipe Template defines the static PI AF Template "OSI_UnitProcedure" for the UnitProcedure level. If missing, the PI AF Template is created and assigned to all UnitProcedure-level event frames regardless of the recipe type.

Recipe Template Example 5:

```
Recipe[2].Template[1].Name    = UP_A
Recipe[2].Template[1].Trigger = [UnitProcedure, value="UProc:1-
*"]
Recipe[2].Template[2].Name    = UP_B
Recipe[2].Template[2].Trigger = [UnitProcedure, value="UProc:2-
*"]
```

In this example, the Recipe Template defines the dynamic PI AF Template for the UnitProcedure-level AF event frame, based on the name of the UnitProcedure. If the AF Template is not found on the AF Server, the interface creates it.

Merging Multiple Source Batches into a Single PIBatch

The Batch interface can merge multiple source batches that have the same BatchID into a single PIBatch. To enable merging, specify the **`/merge`** command line parameter. When a new batch is found on the source, the interface locates the identical batch in the local batch cache and adds the new batch to the existing one. The**`/cachetime`** parameter specifies the duration for which the interface keeps closed batches in the local memory, by default, one day.

Note: The interface only merges batches that are within the cache time frame, during which they are cached in local memory.

If no batch with the BatchID is found, the interface creates one. To configure the interface to use a substring of the source batch BatchID as the BatchID for the new PIBatch, specify the `/bidm` command line parameter. See Using `/BIDM` Parameter for details. Unit batches in a merged batch always contain the original BatchID, Recipe and Product. Each merged batch retains original information such as the full BatchID, Product, Recipe, Formula, and Start and End times, in the PI Properties of the merged batch under the PIProperty node named using the source batch UniqueID.

Configuring Batch IDs

To configure the PI batch ID, specify the **`/bidm`** (**B**atch**ID M**ask) parameter. To define the format of the ID, you specify a list of masks that extract substrings from the value in the BatchID column in the data source. The order of importance depends on the position of the mask in the list. The mask can consist of an array of valid symbols and wildcards.

The following wildcards can be specified in the BatchID mask.

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

The following examples assume that the incoming BatchID column contains "lot30112 / 90dev123 / 12345stp / ld567".

Example	Description	Resulting Batch ID
/bidm=#####	Five contiguous digits and no characters in the substring.	Since there are two matches, the first substring is used and the result is **30112**.
/bidm=######	Six contiguous digits and no characters in the substring. There is no match for this.	The complete string is used as the BatchID.
/bidm=###	Three contiguous digits and no characters in the substring.	Since there are two matches, the first substring is used and the result is **123**.
/bidm=@@@#####	Three contiguous characters followed by five contiguous digits.	**lot30112**.
/bidm=##@@@###	Two digits, three characters, and two more digits.	**90dev123**.
/bidm=#####@@@	Five digits followed by three characters	**12345stp**.
/bidm=?????	Any five characters	**lot30**.

Linking BES Batches to MES Batches

Starting with version 3.0.1.x, the interface enables you to link (bind) Batch Execution Systems batches with Manufacturing Execution System batches. This feature enables you to use PI event frames to build an integrated structure that records the master-slave relationship between the Manufacturing Execution System and its subordinate Batch Execution System. For example, in an environment where an MES System - Werum PAS|X is launching BES Batches (Rockwell FactoryTalk Batch), the FactoryTalk Batch interface always acts as an MES child when configured for lining. For each BES (FactoryTalk) batch, it attempts to find the corresponding parent MES batch event frame that triggered the BES execution. The MES interface always acts as a parent and, if configured for linking, it attempts to find a child BES batch event frame when the event triggering the BES execution is acquired.

Note: This functionality is available only when the interface is configured to write batch data to event frames.

Because the MES and BES interfaces work asynchronously, they require a common element to resolve linkage. To configure this common element, specify the following command line parameter in the startup batch file for both the MES and BES interfaces:

```
/link=<element path>
```

Although an MES batch can be accessed only as the root event frame, the BES batch can be accessed as the root event frame and as the child of the MES event frame that represents the MES processing step that triggered the BES execution.

Loss of Connectivity

The Batch Interface is designed to detect and recover from connection loss from either the data sources or the PI Server, or both. If the connection is lost during processing, the Interface suspends all actions until the PI and data sources are available again. If the data source becomes unavailable, the interface to reconnect on every scan until it succeeds. If the PI server is unavailable, the interface attempts to reconnect every minute (by default) until it succeeds. You can configure the frequency of reconnection attempts and can specify a maximum number of retries. The Interface logs the errors to the local **pipc.log** file.

During normal interface shutdown and startup, no data is lost. All data is buffered by the data sources. If the Interface is interrupted before it can finish processing data from the source to the PI Server, it saves the timestamp of the last good event it processed before shutdown. At startup, the interface resumes processing data starting at the saved timestamp and, after recovering this data, switches to real-time data collection.

Data Preprocessing

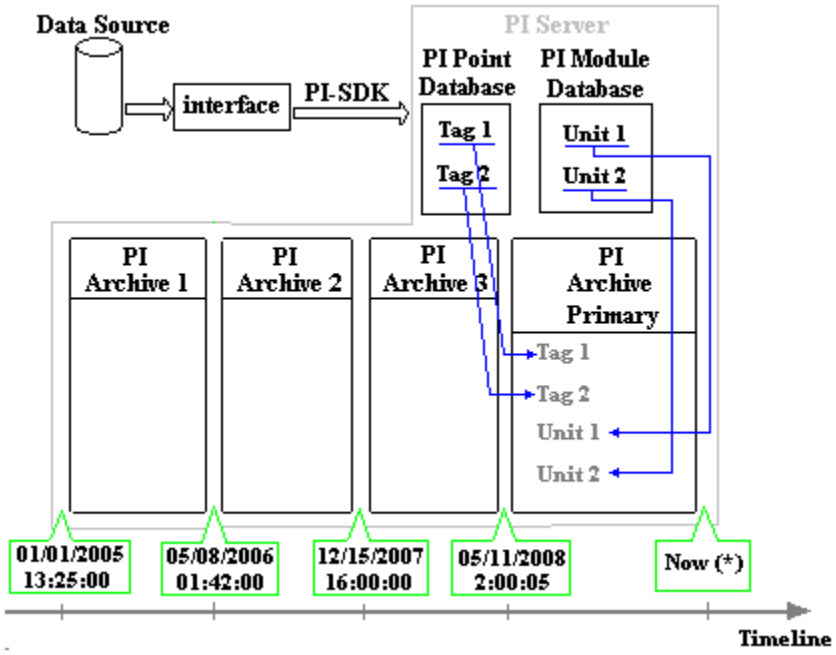
The Batch interface is designed to handle situations when the source data needs to be written to PI archives that are earlier than the primary PI archive. Due to the nature of the PI Server, the newly-added tags, units and modules are indexed (referenced) only in the primary PI archive. Any older archive does not have knowledge of these modules, units and tags. In **Preprocess** mode, the interface creates modules, units, tags and tag aliases but does not process batch data or add events for the tags. After preprocessing, the interface stops and you must reprocess older archives using the offline archive utility.

Note: The PI server does not allow any data to be written to older archives unless the archives contain definitions for the units and tags of interest. Refer to the *PI Server System Management Guide* for details on the archive reprocessing procedure. Note that this procedure has changed in PI Server 2012.

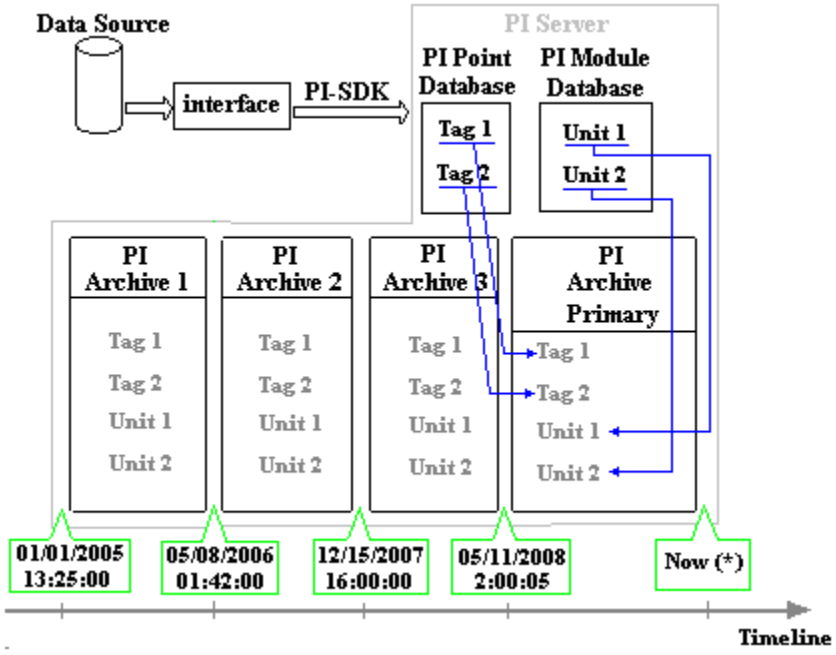
Always run the interface in preprocess mode before writing new batch data to older PI archives. Reprocessing creates indexes for newly-added units, modules, tags in each reprocessed archive.

To run the interface in preprocess mode, specify the **`/mode=noupdate`** parameter with the recovery start time (**`/rst`**) and optional recovery end time (**`/ret`**) parameters.

The following figure illustrates preprocessing for the time range 01/15/2005 12:00:00 to 06/20/2008 13:00:00.



First the interface is run in **Preprocess** mode to create tags and units in the **PIPoint** and **PIModule** databases and references in the **Primary** archive. After reprocessing **PI Archive 1**, **2** and **3** with the PI Archive offline utility (**piarchss**), the **PI archives 1**, **2** and **3** now contain references to the newly-created tags and units as shown in the figure below.



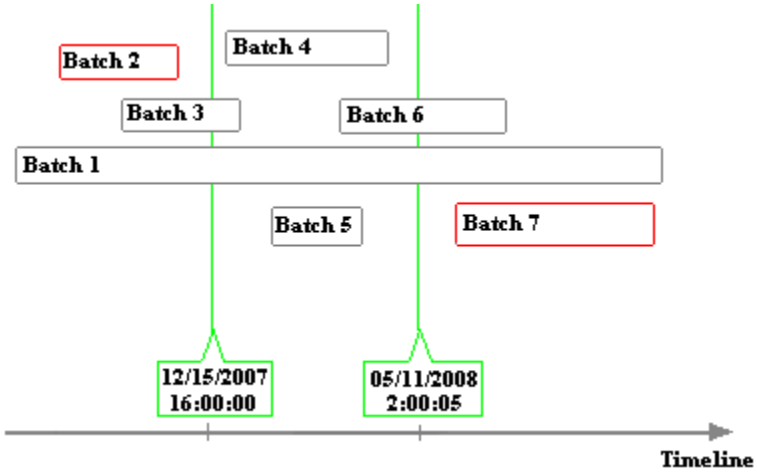
To backfill data into PI Points and the PI Batch database, you can now run the interface in **Recovery** mode.

Data Recovery

The Batch Interface can recover historical data. **Recovery** mode can recover missing data for existing PIModule, PIBatch and PIPoint objects, and it can recover data and create those objects if they do not already exist. These objects include PI modules, PI units, unit-level aliases, phase-level aliases, PIBatches, PIUnitBatches, PISubbatches (Operations, Phases, Phase States and Phase Steps), PIProperties, PIPoints, PIPoint events. When a PI object contains incorrect data (that is, incorrect according to the data source), the interface attempts to correct the PI object to match the data from the source. If the interface cannot correct the, the interface logs an error and you must delete the data and retry recovery.

To enable **Recovery** mode specify a recovery start time and optionally, and end time when starting the interface. If no end time is specified, it prints the results of the recovery process and changes to **RealTime** mode when recovery is complete. If end time is specified, the interface exits after recovery. In **Recovery** mode, open batches are processed only when there are no completed batches to be processed, that is, when the interface reaches the current time.

The following figure illustrates recovery of batches from the data source for the period from 12/15/2007 16:00:00 through 05/11/2008 2:00:05.



To recover these batches, specify the following command line parameters when you start the interface:

```
/rst="12/15/2007 16:00:00" and /ret="05/11/2008 2:00:05".
```

The interface recovers all batches that fall partly (Batch 1, Batch 3 and Batch 6) or entirely (Batch 4 and 5) in the specified period. Batches outside the time frame (Batch 2 and 7) are not recovered.

To recover data from 12/15/2007 16:00:00 until now (*), specify the following command line parameter:

```
/rst="12/15/2007 16:00:00"
```

The interface recovers Batch 7 as well the other batches, then switches to **RealTime** mode.

Data Analysis

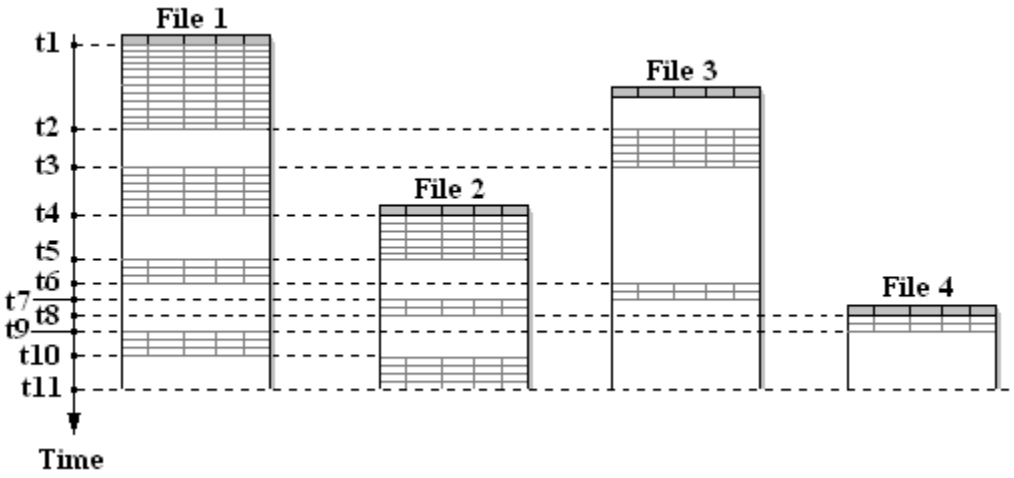
In **Statistics** mode, the Batch interface compares batch data from the data source with batch data in the PI server, logs its results, and exits. To run the interface in **Statistics** mode, specify the **`/mode=stat`** command line parameter, start time (**`/rst`**) and optionally, end time (**`/ret`**) are specified. If you omit end time, the interface analyses data from the specified start time through the current system time.

PI Data Deletion

The Batch interface delete batch data stored in PI server based on the source data. The interface leaves data from all other data sources intact. Delete batch data only if the interface cannot synchronize source data with the PI server in Recovery mode. To delete data, specify the `/mode=delete` command line parameter, plus start time and, optionally, end time for the period of data to be deleted. If you omit end time, data is deleted from the specified start time through the current system time.

EVT Source – Event Based Time Ordered Processing

The Batch interface processes EVT files based on the timestamps of each row within each EVT file, rather than processing EVT files based on names. On each scan, the interface performs a preliminary EVT directory scan to create a time ordered processing queue of all active EVT files based on the current position's timestamp within each file. It then scans each EVT file for the end position. This strategy allows creating a fixed time frame common to all EVT files. The interface will then read data, in the time frame, in time order. Any processing delays due to network losses, server unavailability, slow scan rates, and non alphabetical file naming can be handled gracefully by this approach. The following figure illustrates the difference in alphabetical file name versus event time ordered processing.



There are four EVT files starting at different times and containing data written sequentially by Batch Execution System. The vertical axis is the file time, where the **t1** is the earliest time and the **t11** is the latest time. The alphabetical file name processing sequence is given in the table below.

Order	File Name	Data Segments [start time – end time] written by the BES in parallel EVT files									
		t1 – t2	t2 – t3	t3 – t4	t4 – t5	t5 – t6	t6 – t7	t7 – t8	t8 – t9	t9 – t10	t10 – t11
1	File 1	X		X		X				X	
2	File 2				X			X			X
3	File 3		X				X				
4	File 4								X		

The event time ordered processing is illustrated in the table below

Order	File Name	Data Segments [start time – end time] written by the BES in parallel EVT files									
		t1 – t2	t2 – t3	t3 – t4	t4 – t5	t5 – t6	t6 – t7	t7 – t8	t8 – t9	t9 – t10	t10 – t11
1	File 1	X									
2	File 3		X								
3	File 1			X							
4	File 2				X						
5	File 1					X					
6	File 3						X				
7	File 2							X			
8	File 4								X		
9	File 1									X	
10	File 2										X

Excluding Recipes From Processing

To exclude particular recipes from being processed into the PI Server, specify the **skiprecipes** command in the INI file, specifying the recipes to be skipped as a comma-separated list. You can exclude the following levels: Procedure, UnitProcedure, Operation, and Phase. You can use the following wildcards to specify masks for recipes to be excluded:

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Examples:

```
skiprecipes=recipe1,prc_*nt2
skiprecipes="recipe 1", "prc_paint 2"
```

Double-quote names that contain spaces.

Excluding Units From Processing

To aid in transfer of control between recipes, batches can use "virtual" or "dummy" units that do not physically exist, which can result in overlapping PIUnitBatches and incorrect PIUnitBatch end times. To exclude such units from processing by the interface, specify the **skipunits** command in the INI file command, listing the units to be skipped based on the value encountered in the [UNITID] field of each event. Specify units as a comma-separated list. You can use the following wildcards to specify masks for units to be excluded:

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Examples:

```
skipunits=unit1,u*t2
skipunits="unit 1", "unit 2"
```

Excluding Phases From Processing

To exclude a phase from processing the file, specify the **skipphases** command in the INI file command, listing the phases to be skipped based on the value encountered in the [Phase] field of each event. Specify phases as a comma-separated list. You can use the following wildcards to specify masks for phases to be excluded:

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Examples:

```
skipphases=phase_1,ph*2
skipphases=phase_1, ph*2
```

Excluding Phase States From Processing

To exclude a phase state from processing the file, specify the **excludestates** command in the INI file command, listing the phase states to be skipped based on the value encountered in the [PhaseState] field of each event. Specify phase states as a comma-separated list. You can use the following wildcards to specify masks for phase states to be excluded:

Wildcard	Description
#	Single digit numerical value, 0-9
@	Single alpha character, a-z, A-Z
?	Any single symbol
!	Repeat the previous mask symbol
*	Any set of symbols

Examples:

```
excludestates = COMPLETE, ABO*NG
```

Chapter 3. Installation Checklist

If you are familiar with running PI data collection interface programs, this checklist helps you get the Interface running. If you are not familiar with PI interfaces, return to this section after reading the rest of the manual in detail.

This checklist summarizes the steps for installing this Interface. You need not perform a given task if you have already done so as part of the installation of another interface. For example, you only have to configure one instance of Buffering for every interface that runs on an Interface Node.

The Data Collection Steps below are required. Interface Diagnostics are optional.

Data Collection Steps

1. Confirm that you can use PI SMT to configure the PI Server. You need not run PI SMT on the same computer on which you run this Interface.
2. If you are running the Interface on an Interface Node, edit the PI Server's Trust Table to allow the Interface to write data.
3. Run the installation kit for this Interface. This kit runs the PI SDK installation kit, which installs both the PI API and the PI SDK. This kit also runs the PI Event Frames Interface Manager installation kit, which installs a configuration tool for the interface.
4. If you are running the Interface on an Interface Node, check the computer's time zone properties. An improper time zone configuration can cause the PI Server to reject the data that this Interface writes.
5. Run the PI Event Frames Interface Manager and configure a new instance of this Interface. Essential startup parameters for this Interface are

   Point Source (`/PS=x`)
   Interface ID (`/ID=#`)
   PI Server (`/Host=host:port`)
6. Define sources
7. Important point attributes and their purposes are:

   `Location1` specifies the Interface instance ID.
   `Location2` is the index of the tag
   `Location3` is the point type
   `Location4` specifies the scan class.
   `Location5` is not used by this interface

   `ExDesc` contains the copy of tag name created by the interface.
   `InstrumentTag` is the unit.

8. Start the Interface interactively with command line parameter **/mode=stat** (this mode allows read only from the data sources and PI server) and confirm its successful connection to the PI Server and data sources. If running interface interactively, add switch **/inifile=<full path to INI file>**.

Note: This interface does not use the PI API, therefore PI Buffering (pibufss/bufserv) is not required by the interface.

9. Confirm that the Interface collects data successfully.
10. Configure the Interface to run as a Service. Make sure that interface is **NOT** set as PI Buffer dependent. Confirm that the Interface runs properly as a Service.
11. Restart the Interface Node and confirm that the Interface restarts.

Interface Diagnostics

The interface creates three types of tags that you can use to monitor its performance:

- Health tags
- Object counters
- Timers

All performance tags are named using the interface name as a prefix, with the service ID appended as follows:

`<Prefix> : PIFTBInt_<ServiceID>`

Health Monitoring Tags

There are two health tags: the *heartbeat tag* and the *device status tag*.

The heartbeat tag indicates whether the interface is running. By default, it is updated every 60 seconds. (To configure a higher frequency, specify the `/scan` command line parameter).. The value of the heartbeat tag cycles from 1 to 15.

The device status tag is automatically configured and created if missing by the interface on startup. The following events are written into the device tag:

- `Good` (Indicates that the interface is properly communicating and reading data from the data sources.)
- `1 | Starting”`
- `2 | Connected/No Data | EVT Directory Monitor: <directory name> Initialized.`
- `3 | 1 device(s) in error | Error monitoring directory (onError): <directory name>`
- `3 | 1 device(s) in error | Error monitoring directory: <directory name>`
- `3 | 1 device(s) in error | Failed to start directory monitoring thread: <directory name>`

- `3 | 1 device(s) in error | Error in scanning directory: <directory name>`
- `3 | 1 device(s) in error | Error obtaining EVT files EOF.`
- `3 | 1 device(s) in error | Error getting current EVT file timestamp.`
- `3 | 1 device(s) in error | Error reading EVT file: <filename>.`
- `3 | 1 device(s) in error | Error while reading EVT file.`

Object Counters

Object counter tags monitor the number of different type objects read from the source and written to the PI server. These tags are automatically created when interface first starts up, and zeroed every time the interface is restarted thereafter. Archiving flag for these tags is turned off. The following counter tags are provided.

Tag Name	Loc3	ExcDesc	Description
<Prefix>_EventReadCount	2	[UI_EVENTREADCOUNT]	Number of events read from the source since last startup.
<Prefix>_ErrorCount	3	[UI_ERRORCOUNT]	Number of errors occurred since last startup.
<Prefix>_SourceUnitCount	4	[UI_SOURCEUNITCOUNT]	Number of Units found on the data source(s) since startup.
<Prefix>_PIUnitCount	5	[UI_PIUNITCOUNT]	Number of Units found and added on the PI server since startup.
<Prefix>_SourcePhaseModCount	6	[UI_SOURCEPHASEMODCOUNT]	Number of Phase Module found on the data source(s) since startup.
<Prefix>_PIPhaseModCount	7	[UI_PIPHASEMODCOUNT]	Number of Phase Module found and added on the PI Server since startup.
<Prefix>_SourceBatchCount	8	[UI_SOURCEBATCHCOUNT]	Number of batches found on the data source(s) since startup.
<Prefix>_PIBatchCount	9	[UI_PIBATCHCOUNT]	Number of PIBatch objects found and added on the PI server since startup.
<Prefix>_SourceUnitBatchCount	10	[UI_SOURCEUNITBATCHCOUNT]	Number of unitbatches found on the data sources(s) since startup.
<Prefix>_PIUnitBatchCount	11	[UI_PIUNITBATCHCOUNT]	Number of PIUnitBatch objects found and added on the PI server since startup.
<Prefix>_SourceSubBatchCount	12	[UI_SOURCESUBBATCHCOUNT]	Total number of operations+phases+phase states found on the data source since startup.
<Prefix>_PISubBatchCount	13	[UI_PISUBBATCHCOUNT]	Total number of PISubBatch objects founded and added to the PI server since last startup.

Tag Name	Loc3	ExcDesc	Description
<Prefix>_SourcePropertyNodeCount	14	[UI_SOURCEPROPNODECOUNT]	Number of property nodes found in data source(s) since last startup
<Prefix>_PIPropertyNodeCount	15	[UI_PIPROPNODECOUNT]	Number of PIProperty objects (nodes) found and added to the PI server since last startup.
<Prefix>_SourcePropertyEventCount	16	[UI_SOURCEPROPEVENTCOUNT]	Number of events to be written to the batch properties found on the data source(s) since last startup.
<Prefix>_PIPropertyEventCount	17	[UI_PIPROPEVENTCOUNT]	Number of PIProperties(events) found and added to the PI server since last startup.
<Prefix>_SourceTagCount	18	[UI_SOURCETAGCOUNT]	Number of tags found on the data source(s) since last startup
<Prefix>_PITagCount	19	[UI_PITAGCOUNT]	Number of PIPoints found and added to the PI server since last startup.
<Prefix>_SourceTagEventCount	20	[UI_SOURCETAGEVENTCOUNT]	Number of events to be written into tags found on the data sources(s) since last startup.
<Prefix>_PITagEventCount	21	[UI_PITAGEVENTCOUNT]	Number of events written into PIPoints on the PI server since last startup.
<Prefix>_SourceTagAliasCount	22	[UI_SOURCETAGALIASCOUNT]	Number of tag aliases to be created based on the data source(s) since last startup.
<Prefix>_PITagAliasCount	23	[UI_PITAGALIASCOUNT]	Number of PIAliases found and added to the PI server since last startup.
<Prefix>_CachedBatchCount	24	[UI_CACHEDBATCHCOUNT]	Number of batch objects cached in the local memory.
<Prefix>_OpenBatchCount	25	[UI_OPENBATCHCOUNT]	Subset of cached objects which still have no end time set.
<Prefix>_WaitingForEquipmentUB	34	[UI_UBWAITFOREQUIP]	Number of UnitBatches which do not have equipment allocated yet. The allocation is check at PI Server synchronization routine.

Timers

Timer tags report how long per scan it took the interface to read the data source, cache local data and synchronize cached data with PI server. The following timer tags are provided.

Tag Name	Loc3	ExcDesc	Description
<Prefix>_SourceReadTime	26	[UI_SOURCEREADTIME]	The time per scan it took the interface to read data from data source(s).
<Prefix>_TagCacheTime	27	[UI_TAGCACHETIME]	The time per scan it took the interface to populate local tag cache.

Tag Name	Loc3	ExcDesc	Description
<Prefix>_BatchCacheTime	28	[UI_BATCHCACHETIME]	The time per scan it took the interface to populate the local batch cache.
<Prefix>_EquipmentCacheTime	29	[UI_EQUIPCACHETIME]	The time per scan it took the interface to populate the local equipment (module) cache.
<Prefix>_BatchSyncTime	30	[UI_BATCHSYNCTIME]	The time per scan it took the interface to synchronize local batch cache with the PI server.
<Prefix>_TagSyncTime	31	[UI_TAGSYNCTIME]	The time per scan it took the interface to synchronize local tag cache with the PI server.
<Prefix>_EquipmentSyncTime	32	[UI_EQUIPSYNCTIME]	The time per scan it took the interface to synchronize local equipment cache with the PI server.
<Prefix>_TotalTime	33	[UI_TOTALTIME]	The total time per scan it took the interface to read data, cache it in the local memory and synchronize local cache wit PI server.

Chapter 4. Interface Installation

OSIsoft recommends that interfaces be installed on PI Interface Nodes instead of directly on the PI Server node. A PI Interface Node is any node other than the PI Server node where the PI Software Development Kit (PI SDK) has been installed (see the PI SDK manual). With this approach, the PI Server need not compete with interfaces for the machine's resources. The primary function of the PI Server is to archive data and to service clients that request data.

Note: Buffering is not recommended with the PI Interface for FactoryTalk Batch. This is due to the fact that the source data is already effectively buffered on the source.

In most cases, interfaces on PI SDK nodes should be installed as automatic services. Services keep running after you logs off. Automatic services automatically restart when the computer is restarted, which is useful in the event of a power failure.

The guidelines are different if an interface is installed on the PI Server node. In this case, the typical procedure is to install the PI Server as an automatic service and interfaces as manual services that are launched by site-specific command files when the PI Server is started. Interfaces that are started as manual services are also stopped in conjunction with the PI Server by site-specific command files. This typical scenario assumes that Bufserv is not enabled on the PI Server node. Bufserv can be enabled on the PI Server node so that interfaces on the PI Server node do not need to be started and stopped in conjunction with PI, but it is not standard practice to enable buffering on the PI Server node.

Naming Conventions and Requirements

In the installation procedure below, it is assumed that the name of the interface executable is `PIFTBInt.exe`, the startup command file is called `PIFTBInt.bat`, and the initialization file is called `PIFTBInt.ini`.

When Configuring the Interface Manually

When configuring the interface manually it is customary for you to rename the executable, the startup command and initialization files when multiple copies of the interface are run. For example, `PIFTBInt1.exe`, `PIFTBInt1.bat` and `PIFTBInt1.ini` would typically be used for interface number 1, `PIFTBInt2.exe`, `PIFTBInt2.bat` and `PIFTBInt2.ini` for interface number 2, and so on. When an interface is run as a service, the executable and the command file must have the same root name because the service looks for its command-line parameters in a file that has the same root name.

Interface Directories

Interface Installation Directory

The interface install kit will automatically install the interface to:

`PIHOME\Interfaces\FTBInt\`

`PIHOME` is defined in the `pipc.ini` file.

Interface Installation Procedure

To install, run the appropriate installation kit.

`FTBInt_#.#.#.#.exe`

Installing the Interface as a Windows Service

The Batch interface service can be created with the PI Event Frames Interface Manager or can be created manually.

Installing the Interface Service with the PI Event Frames Interface Manager

The PI Event Frames Interface Manager provides a user interface for creating, editing, and deleting the interface service on the **Service** tab.

Service Tab

Settings for installing and running the interface as a Windows Service.

Setting	Description
Display Name	The name of service as displayed in the Services control Panel. To indicate that the service is part of the OSIsoft suite of products, prefix the name with “PI”.
Logon as	The Windows user account used to run the interface service.
Password	Password for the Windows user account used to run the interface service.
Startup Type	Configures whether the service starts automatically when the interface node is rebooted.
Dependencies	Configures other services that the Batch interface requires in order to run.

Installing the Interface Service Manually

Help for installing the interface as a service is available at any time with the command:
`PIFTBInt.exe -help`

Change to the directory where the **`PIFTBInt.`**`exe` executable is located. Then, consult the following table to determine the appropriate service installation command.

Windows Service Installation Commands on a PI Interface Node or a PI Server Node without Bufserv implemented	
Manual service	PIFTBInt.exe -install -depend tcpip
Automatic service	PIFTBInt.exe -install -auto -depend tcpip

*Automatic service with service id	PIFTBInt.exe -serviceid X -install -auto -depend tcpip

*When specifying service id, you must include an id number. It is suggested that this number correspond to the interface id (**/id**) parameter found in the interface .bat file.

Check the Microsoft Windows services control panel to verify that the service was added successfully. The services control panel can be used at any time to change the interface from an automatic service to a manual service or vice versa.

Chapter 5. Digital States

For more information regarding Digital States, refer to the PI Server documentation.

Digital State Sets

PI digital states are discrete values represented by strings. These strings are organized in PI as digital state sets. Each digital state set is a user-defined list of strings, enumerated from 0 to n to represent different values of discrete data. For more information about PI digital tags and editing digital state sets, see the PI Server manuals.

An interface point that contains discrete data can be stored in PI as a digital tag. A Digital tag associates discrete data with a digital state set, as specified by you.

System Digital State Set

Similar to digital state sets is the system digital state set. This set is used for all tags, regardless of type to indicate the state of a tag at a particular time. For example, if the interface receives bad data from an interface point, it writes the system digital state `bad input` to PI instead of a value. The system digital state set has many unused states that can be used by the interface and other PI clients. Digital States 193-320 are reserved for OSIsoft applications.

Chapter 6. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For example, the string **EV** may be used to identify points that belong to the Batch Interface. To implement this, the PointSource attribute would be set to **EV** for every PI Point that is configured for the Batch Interface. Then, if **/ps=EV** is used on the startup command-line of the Batch Interface, the Interface will search the PI Point Database upon startup for every PI point that is configured with a PointSource of **EV.** Before an interface loads a point, the interface usually performs further checks by examining additional PI point attributes to determine whether a particular point is valid for the interface. For additional information, see the **/ps** parameter.

Case-sensitivity for PointSource Attributes

The PointSource character that is supplied with the **/ps** command-line parameter is not case sensitive. That is, **/ps=P** and **/ps=p** are equivalent.

Reserved Point Sources

Several subsystems and applications that ship with the PI are associated with default PointSource characters. The Totalizer Subsystem uses the PointSource character `T`, the Alarm Subsystem uses `G` and `@`, Random uses `R`, RampSoak uses `9`, and the Performance Equations Subsystem uses `C`. Do not use these PointSource characters or change the default point source characters for these applications. Also, if a PointSource character is not explicitly defined when creating a PI point; the point is assigned a default PointSource character of `Lab` (PI 3). Therefore, it would be confusing to use `Lab` as the PointSource character for an interface.

Note: Do not use a point source character that is already associated with another interface program. However it is acceptable to use the same point source for multiple instances of an interface.

Chapter 7. PI Point Configuration

The PI point is the basic building block for controlling data flow to and from the PI Server. The batch interface automatically builds all points based on the information found in INI file.

Process parameters are often specified in batch data sources. These parameters are typically more easily viewed as a graphical trend. Points may be built to specify which events are to be captured and stored in PI. Please refer to section **Event Logging - Tag Template** for information on how to configure Tag Templates for specific event capturing**.**

Chapter 8. Startup Command File

Command-line parameters can begin with a **/** or with a **-**. For example, the **/ps=E** and **-ps=E** command-line parameters are equivalent.

For Windows, command file names have a .bat extension. The Windows continuation character (**^**) allows for the use of multiple lines for the startup command. The maximum length of each line is 1024 characters (1 kilobyte). The number of parameters is unlimited, and the maximum length of each parameter is 1024 characters.

Configuring the Interface with PI Event Frames Interface Manager

The PI Event Frames Interface Manager provides a graphical user interface for configuring the interface. If the interface is configured with this tool, the batch file of the interface and the interface settings file will be created and maintained by the PI Event Frames Interface Manager and all configuration changes will be kept in that file. The procedure below describes configuration using PI Event Frames Interface Manager to configure the Batch Interface.

Interface Selection Tab

To create a new instance of the interface, perform the following steps:

1. Click **Add Interface**. A browse dialog is displayed.
2. Browse to the interface installation directory, select the executable for the interface, and click **Open** to dismiss the browse dialog.
3. Click **OK**. The **Interface** field displays the name of the interface instance you created.

The PI Event Frames Interface Manager can be used to manage multiple instances of the interface.

File Selection Tab

The **File Selection** tab is used to select the interface settings file that stores settings and the configuration for the interface instance:

- Interface Settings File (.INI): Contains the interface startup parameter and configuration settings. Be sure to specify the .ini extension.

Server Information Tab

The **Server Information Tab** is where you specify the PI Server and PI AF Server systems that you intend to use with the interface instance. The interface stores data in PI Tags on a PI Server. It can generate either batches in the PI Server Batch Database or event frames on a PI AF Server.

PI Server (/HOST)

Specifies the PI Server node to which the interface writes PI tag data. **Host** is the IP address of the PI Sever node or the fully qualified domain name of the PI Server node. If the PI Server you want to use is not in the drop down list, you must add it to the known servers table using the **AboutPI-SDK** application.

[PI Server] User and [PI Server] Password

For PI Servers version 3.4.380.36 and higher, use Windows Integrated Security for authentication. Omit you name and password from these fields, and ensure that the Windows account that runs the interface has sufficient permissions on the PI Server to write data to PI Points.

For PI Servers prior to version 3.4.380.36, configure a trust on the PI Server that permits access for you running the interface or interface service.

[PI Server] Port

The port number for TCP/IP communication. The default port (recommended) is 5450.

Use PI AF server

Check this box to create event frames on a PI AF server, instead of creating batches in the PI Server Batch Database.

[PI AF] Host and Database (/AFHOST and /AFDATABASE)

The destination PI AF server node and database where you want the interface to create event frames.

[PI AF] User and [PI AF] Password

If you are not using Windows Integrated Security for authentication (recommended), enter you name and password for the Windows user account that you intend to use to connect to PI AF.

Source

On this tab, you define the data sources from which the interface will read data. The interface can read data from multiple data sources. To read from EVT file, add an EVT source, and specify the directory where the EVT files are.

Filters Tab

This tab configures the Recipes, Units, Phases, or Phase States that should be excluded from processing by the Batch Interface.

Skip Phases (/SKIPPHASES)

The interface will not process any event with the listed phases in the [Phase] or [Phasemodule] column.

Skip Units (/SKIPUNITS)

The interface will not process any event with the listed units in the [Unit] column.

Skip Recipes (/SKIPRECIPES)

The interface will not process any event with the listed recipes in the appropriate column ([Procedure] for a procedure recipe, [UnitProcedure] for a unitprocedure recipe, etc.)

Exclude Phase States (/EXCLUDESTATES)

The interface will not write phase state events to the PI system with the listed phase state in the [PhaseState] column.

Time Settings Tab

This tab configures the time settings that control how the Batch Interface handles server connections and processes data.

Query Time Settings

Scan (/SCAN=<seconds>)

Specifies how frequently the interface scans the data source.

Cache time (/CACHETIME=<days>)

Specifies how long completed events are retained in memory. Default is one day. Specify the maximum duration expected between event frames that need to be merged, plus any desired margin of safety.

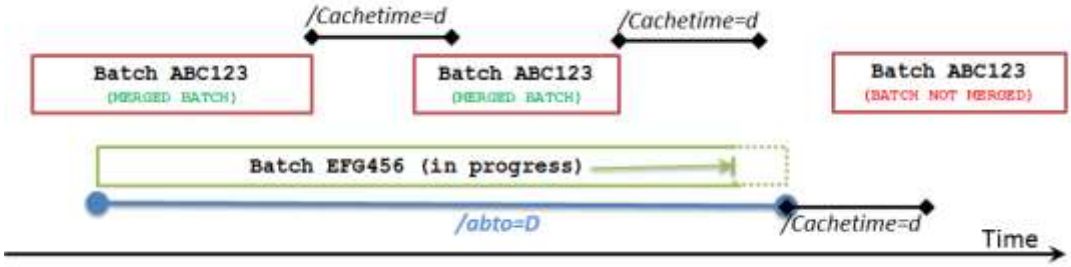
The value can be specified as whole day or fraction of a day. For example, to release completed batches when their end time is less than 7 days and 12 hours from current time, specify the following cache time setting:

```
CACHETIME=7.5 days
```

Abandoned batch timeout (/ABTO=<days>)

Specifies how long event frames can remain open before being considered abandoned. When this period (plus cache time) elapses, the interface writes an end time to the event to close it. Specify the maximum duration expected for events, plus any desired margin of safety.

For example, if you set abandoned batch timeout to 50.5 days and cache time is seven days, events open for 57.5 days are automatically closed. The following figure illustrates timeout logic.



Maximum query time frame (/MAXQTF=<days>)

To help manage workload and memory usage, defines the maximum time frame for queries. For example, if you specify 30 days and the interface queries for one year's worth of data, the interface issues 12 one-month queries rather than one (large) one-year query.

Maximum stop time (/MAXSTOPTIME=<seconds>)

Specifies the maximum time allowed for the interface to properly shutdown. If the shutdown process takes longer than the specified time, the interface is forced to terminate immediately. The default value is 120 seconds.

Use local time stamps to process incoming events (/TS)

Applies the time on the local machine to events from the datasource. By default, timestamps are recorded using GMT.

PI Connection Settings

PI connection timeout (/PICONNTO=<seconds>)

Override the default PI SDK Connection TimeOut property.

PI data access timeout (/PIDATO=<seconds>)

Override the default PI SDK Data Access TimeOut property.

Retry (/RETRY=<seconds>)

Specifies how long to wait before retrying a failed SDK attempt to write data to PI Server. The default is 60 seconds.

Retry timeout (/RETRYTO=<seconds>)

Specifies timeout for failed SDK attempts to write data to PI Server. To avoid data loss, set to 0 (default, no timeout).

SQL server Connection Settings

SQL connection timeout (/SQLCONNTO=<seconds>)

Override the default SQL connection timeout. The default is 60 seconds.

SQL data access timeout (/SQLDATO=<seconds>)

Override the default SQL data access timeout. The default is 100 seconds.

Operational Settings Tab

The settings on this tab configure the mode in which the Batch Interface runs and other, related settings.

Runtime mode (/MODE=<mode>)

Interface modes are as follows:

Mode	/MODE Flag	Description
Realtime (default)	REALTIME	Scan data source to collect data in realtime
Recovery	REALTIME with /RST specified	Scan data source and generate or correct events accordingly. The Batch Interface always starts in recovery mode, then switches to realtime mode.
Statistics	STAT	Compare data source history against events and report results without updating any data.
Delete	DELETE with /RST and /RET specified	Delete events for a specified period.

Perform one scan then stop (/SINGLERUN)

The interface performs one scan of active points, then exits.

Print result of first scan to file (/PRINT=<file name>)

The name of the text file to which the results of the first scan are printed. The results include the event frame hierarchy tree, the tag list, and the equipment tree. This

parameter is designed primarily for troubleshooting and configuration testing when the interface is run in statistics mode.

Debug level (/DB=<#>)

Specifies level of detail for logging as follows:

- 0: Log errors and warnings (default)
- 1: Log errors, warnings and major successes
- 2: Verbose logging

Numeric settings (/NS=<lang>)

Configures how numeric values are formatted by the interface, to enable the interface to properly interpret numeric values based on the machine's regional setting or a user-specified language. Default is "English_UnitedStates".

Interface ID (/ID=x)

Specifies the numeric interface instance identifier (maximum nine digits). To detect PI points maintained by the interface instance, the interface matches this setting against the value in the points' `Location1` attribute.

Point source (/PS=x)

Point source for the interface instance. Point source is not case sensitive. Corresponds to the `PointSource` attribute of individual PI Points. The interface loads PI points with the same point source.

Associate all reference elements with child Event Frames (/DPRETC)

When creating Event Frames in PI AF, by default all reference elements are associated with child event frames.

Failover Settings

Failover tag (/FAILOVERTAG=x)

The tag on the PI Server that will be used to coordinate failover.

Failover identifier (/FAILOVERID=x)

The unique identifier of this interface in failover.

Failover swap time (/SWAPTIME=<seconds>)

How long an interface should be inactive before another interface will assume writing data to the PI system.

Security Settings

Specify point security (/PTSEC=x)

Override the default ptsecurity of PIPoints created by the interface.

Specify data security (/DATASEC=x)

Override the default datasecurity of PIPoints created by the interface.

Save Settings Tab

Saves the configuration.

Test Configuration Tab

Tests the configuration settings. Specify test settings as follows, click **Run Test**, then check the output file for results.

Field	Description
`BAT File`	The path to the interface .BAT file to run during the test execution.
`Output File`	The path to the text file where the test results are written.
`Start Time`	Start time for scanning.
`End Time`	End time for scanning

Configuring Interface Startup Files

The interface has two startup configuration files; `PIWPASXBatch.bat` and `PIFTBInt.ini`. The .bat file is required and is the primary file for specifying interface configurations. The INI file is used to specify the interface configurations, such as data sources, translations, product template, equipment template, tag templates and property templates.

When using the .INI file, each parameter should be defined on separate line. There should be only one equal (=) sign per line. Parameters can be disabled by prefixing the parameter lines with two forward slashes (//)

When configuring the .bat startup file the continuation character ^ can be used to allow multiple lines for defining parameters. The maximum length for a single line is 1024 characters (1 kilobyte). This is a Windows limitation.

Command-line Parameters

This is a listing of the command-line parameters and their specific behavior with respect to the PI Batch interface. This section gives more detailed information concerning the parameters that may be specified when configuring the interface.

Parameter	Description
/abto=<days> Optional Default: 100 days	(**A**)Bandoned (**B**)atch (T)ime(**O**)ut. Defines the time period from the cached batches time frame into the past after which the open batches are considered to be abandoned and can be released from the interface's local cache. The default value is 100 days. Example: If **/abto=50.5** and **/cachetime=7.1** then the batches with last event occurred before NOW() – 7.1 days – 50.5 days will be considered abandoned and removed from the local interface memory. --|----------[-cached batches time frame -] ---> Timeline -57.6 days -7.1 days (current time)
/bidm=<list> Optional	The **/bidm** switch (**B**atch **ID M**ask) is used to obtain a new BatchID, which is a substring of the value in the source BatchID field. The **/bidm** takes a list of masks as the input value. Each BatchID mask can consist of an array of valid symbols and wildcards. The following wildcards are supported the interface: # - single digit numerical value (0-9) @ - single alpha character (a-z, A-Z) ? – any single valid symbol ! – repeat previous BatchID mask symbol * - any array of ? symbols. Example: Let's say that the BatchID column in the event file is **lot30112 / 90dev123 / 12345stp / ld567**. The **/bidm="#####"** will result in new BatchID **30112**. The **/bidm="##@!"** will result in new BatchID **90dev**. The **/bidm="*##@!"** will result in new BatchID **lot30112 / 90dev**. The **/bidm="@@@@, #8dev4, #!"** will result in new BatchID **30112**. Since the first and second masks could not be found, third mask is used instead.
/CacheTime=<days> Optional Default: 1 day	Defines the time period for which the completed batches are retained in the memory. [(*-cachetime) - *] The default value is 1.0 day. The value can be specified as whole day or fraction of the day. Example: **/cachetime=7.5** days In this case the interface is going to release completed batches when their end time is going to be less than 7 days and 12 hours from current time.

Parameter	Description
/dac Optional	The /dac Disable Arbitration Counters parameter informs interface to release unit on the first Resource Release event even though the number of Acquire events is higher than number of Release events. By default, interface requires number of Resource Release events to be the same as Resource Acquire events for each unit to release the unit.
/DataSec=<string> Optional	The **/DataSec** parameter specifies the PIPoint Data Access Security rights. These rights are assigned to interface-generated tags during point creation. This string has different forms. If PIPoints are created on a PI Server 3.4.375.99 or earlier, it will have an owner, group, world format. Example: /datasec="**o:rw g:r w:r**" If PIPoints are created on a PI Server 3.4.380.36 or later, it must specify an Access Control List (ACL) Example: /datasec="piadmin: A(r,w) | PIEngineers: A(r)"
/db=[#] Optional Default: 0	The **/debug=[#]** parameter specifies the Interface debug logging message level. There are three levels that may be assigned: 0 – Log only errors and warnings. 1 – Log errors, warnings and major success messages 2 – Log ALL messages. Log level two (2) is the most verbose setting; while level zero (0) reports the least detail (it logs only error messages). The default logging level is **0**, to log errors and warnings only. When testing the Interface, it may be necessary to use a more verbose setting (1 or 2).
/dpretc Optional AF Only	By default, the interface propagates each event frame element reference to its children event frames. This functionality can be disabled by specifying the following command line parameter: /dpretc - (D)isable (P)ropagation of (R)eferenced (E)lements (T)o (C)hilren.
/FailOverID=<string> Optional	Configure the unique failover ID for the interface instance. Must be used with the `/FailOverTag` parameter. Example: /FailOverID="intf1"
/FailOverTag=<PI Point Name> Optional	Specifies the PI point that is used to track which interface instance is primary. Must be used with the /FailOverID parameter. Example: /FailOverTag="Batch_FailoverTag"
/host=host:port	The **/host** parameter is used to specify the destination PI

Parameter	Description
Required	server node where the data is going to be stored. `Host` is the IP address of the PI Sever node or the domain name of the PI Server node. **`Port`** is the port number for TCP/IP communication. The port is always 5450. It is recommended to explicitly define the host and port on the command-line with the `/host` parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults. Examples: The interface is running on a PI Interface Node, the domain name of the PI home node is Marvin, and the IP address of Marvin is 206.79.198.30. Valid **`/host`** parameters would be: **`/host=marvin`** **`/host=marvin:5450`** **`/host=206.79.198.30`** **`/host=206.79.198.30:5450`**
`/id=x` Required	The **`/id`** parameter is used to specify the interface identifier. The interface identifier is a string that is no longer than 9 characters in length. This interface uses the **`/id`** parameter to identify a particular interface copy number that corresponds to an integer value that is assigned to one of the Location code point attributes, most frequently Location1. For this interface, use only numeric characters in the identifier. For example, **`/id=1`**
`/IniFile=<UNC Path>` `Optional`	This parameter allows you to specify an alternate Path and Filename for the INI file. If not specified, the interface will expect to find the INI file in the same directory as, and expect the INI file to have the same file name (but with an .INI extension) as the interface executable.
`/link=<Element Path>` **`Optional`** **`AF Only`**	Link BES (FactoryTalk) batches to MES (Werum PAS-X) batches. Available only when the interface is configured to write batch data to AF Server event frames. When configured thus, the FactoryTalk Batch interface always acts as MES child. For each BES (FactoryTalk) batch, it attempts to find the corresponding parent MES batch event frame that triggered the BES execution. Similarly, the MES interface always acts as a parent and, if configured for linking, it always attempts to find a child BES batch event frame when the BES execution event is acquired. Note: While MES batch can be accessed only as the root event frame, the BES batch can be accessed as the root event frame and as the child of the MES event frame that represents the MES Basic Function that triggered the BES execution. Example: FactoryTalk interface must contain command line parameter: /link=MESCommon WPASX interface must contain the same command line parameter: /link=MESCommon

Parameter	Description
/MaxStopTime=<seconds> Optional Default: `120 seconds`	The **/maxstoptime** parameter is used to set the maximum time allowed for the Interface to properly shutdown. The value must be given in seconds. If the Interface shutdown process takes longer than the specified time, the Interface will be forced to terminate immediately.
/MaxQTF=<days> Optional	Maximum Query Time Frame. This parameter sets the maximum time frame for each query made to source. The value can be fractional. Valid range of values: 0.001 to 180 Default: 30 Example: if **/rst=01/01/2005** and **/ret=12/30/2007**, then the actual data time frame to be processed is ~2 years. This can be very memory intensive and potential run out of memory. With the help of this parameter, the interface is going to break 2year query into smaller sub queries with time frame = 30 days each by default. So the actual queries will be performed with the following time frames: [01/01/2005 – 31/01/2005] [31/01/2005 – 02/03/2005] [02/03/2005 - 01/04/2005] Etc.

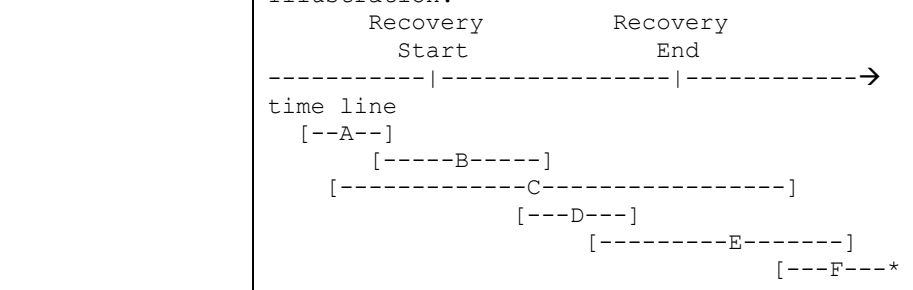
Parameter	Description
/Merge Optional	The **/merge** switch allows the interface to merge multiple source batches with same BatchID into one PIBatch. Original data for each merged batch is stored in PIProperties under PI Property Node named as UniqueID of the original batch. This data includes: original BatchID, StartTime (UTC), EndTime(UTC), Product and Formula Name. Merging time frame is controlled by **/cachetime** switch, i.e. the interface will only merge batches which are still cached in local memory. **Note:** If BatchID's are different, use additional switch **/bidm**. This switch allows to identify common subset of characters in BatchID and then merging will be performed based on this subset in addition to actual BatchID merging. Example: There are 5 running batches within **/cachetime** timeframe: Test12345_1, Test_12345_2, CleaningTest, USPO12345_test, CleaningTest With **/merge** switch defined: there will be: 4 separate batches: Test12345_1, Test_12345_2, USPO12345_test And 1 merged batch: CleaningTest With additional **/bidm=#####** switch defined, where # is the wildcard for numerical values. There will be only 2 merged batches. Note: the unitbatches will have its original BatchID's: Batch(1): 12345 UnitBatches: Test_12345_1 Test_12345_2 Test_12345_test Batch(2): CleaningTest UnitBatches: CleaningTest CleaningTest Equivalent to Recipe Template definition in INI file: Recipe[1].Merge = true

Parameter	Description
/Mode=<mode> Optional Default: `Normal` `Possible values:` **/Mode=normal** (or no switch defined) **/Mode=delete** **/Mode=stat** **/Mode=nodata**	The **/Mode** parameter is used to set the running mode of the Interface. There are four available modes: **Normal** – (default) The Interface will perform realtime processing. This mode is also used for historical data recovery. To activate recovery mode, **/rst** switch has to be defined in command line parameters. In Recovery mode, if the /ret switch was not defined; the interface is going to recover data until current time, then switch to realtime processing automatically. If **/ret** switch was defined, then the interface is going to stop on completion of recovery process. **Stat** – In this mode, the interface only compares source data with the PI server data. Note, the interface does not write or modify any data on the PI Server. On completion the interface reports results and stops. **Delete** – In this mode the interface cleans PI archives based on specified source data only, leaving data from all other sources intact. This mode should be used only if the interface is unable to synchronize source batch data with the PI server. This modes is used only in conjunction with Recovery mode switches (**/rst** and **/ret**). **NoData** – This mode is designed for situations when the source data needed to be written to PI archives which are earlier than the primary PI archive. Due to the nature of the PI Server, the newly added tags, units and modules are indexed (referenced) only in the primary PI archive. Any older archive will not have any knowledge of these modules, units and tags. In **/Mode=NoData** the interface creates only modules, units, tags and tag aliases without processing batch data and pushing events into the tags. On completion, the interface stops and you has to reprocess older archives with offline archive utility. The manual archive reprocessing creates indexes for newly added units, modules, tags in each reprocessed archive. This mode should be always used before writing new batch data to older PI archives (other than Primary).
/mop Optional	The /mop Merge Operation switch allows to combine same named operations running under the same UnitProcedure into a single operation. The start time of the combined operation is the start of the earliest operation and the end time is the end time of the latest/ longest operation which was merged. Equivalent to Recipe Template definition in INI file: Recipe[3].Merge = true
/mup Optional	The /mup Merge Unit Procedures switch allows to combine sequential multiple Unit Procedures with the name and running on the same unit into a single UnitProcedure. The merge will not occur if the Unit of interest was used by another recipe between candidates for merging. The start time of the combined Unit Procedure is the start of the earliest Unit Procedure and the end time is the end time of the latest/ longest Unit Procedure which was merged. Equivalent to Recipe Template definition in INI file: Recipe[2].Merge = true

Parameter	Description
/noarbitration	This switch is used when the source Batch Executive System (BES) provides batch data without equipment arbitration. When this switch used, PI UnitBatches are created based on source batch recipe data only.
/ns=[lang] Optional	The **/ns** (**N**umeric **S**ettings) switch allows the interface to perform proper numerical conversions based on the "Regional and Language Options" setting on local system or based on user defined language. This switch is particularly useful when the numerical conventions differ (example a comma is used instead of a decimal etc) from the default settings. If the switch is not used, then the default settings of "English_UnitedStates" is used. If the switch is used without any language specification, i.e. **/ns**, then the interface will use "Regional and Language Options" settings specified on the Windows machine where the interface is running. If the language specification is passed as a value (**/ns**=lang), then the interface will use that value as internal regional/language setting to perform numerical conversions regardless of local system "Regional and Language Options" setting. If the switch contains invalid language, .i.e **/ns=<invalid language>**, then the interface will exit. The language can be passed by type as it is specified below or by its abbreviation. Language types (abbreviations): chinese chinese-simplified (chs) chinese-traditional (cht) czech (csy) danish (dan) belgian, dutch-belgian (nlb) dutch (nld) australian, english-aus (ena) canadian, english-can (enc) english english-nz (enz) english-uk (uk) american, american-english, english-american, english-us, english-usa, (enu) (us) (usa) finnish (fin) french-belgian (frb) french-canadian (frc) french (fra) french-swiss (frs) german-swiss, swiss (des) german (deu) gegerman-austrian (dea) greek (ell) hungarian (hun) icelandic (isl)

Parameter	Description
	italian (ita) italian-swiss (its) japanese (jpn) korean (kor) norwegian-bokmal (nor) norwegian norwegian-nynorsk (non) polish (plk) portuguese-brazilian (ptb) portuguese (ptg) russian (rus) slovak (sky) spanish (esp) spanish-mexican (esm) spanish-modern (esn) swedish (sve) turkish (trk) Examples: **/ns -** will set the interface to use the local Windows language/regional settings **/ns=italian** **/ns=ita** Both switches will set the interface to use Italian language/regional settings.
/PIConnTO=<seconds> Optional	This parameter is used to change the current **PI Conn**ection **T**ime**O**ut property. By default the Interface uses the default SDK settings.
/PIDATO=<seconds> Optional	This parameter is used to change the current **PI D**ata **A**ccess **T**ime**O**ut property. . By default the Interface uses the default SDK settings.
/PIPswd=<password> Optional Default: Use Trust Table	The **/PIPswd** parameter is used to explicitly specify you password to establish the connection to the PI Server. If this parameter is not specified, the Interface will try to use the trust table. Note: The **/PIPswd** parameter must be used in conjunction with the **/PIUser** parameter.
/PIUser=<name> Optional Default: Use Trust Table	The **/PIUser** parameter is used to explicitly specify you name to establish a connection to the PI Server. If this parameter is not specified, the Interface will try to use the trust table.
/Print=<file name> Optional	Prints the results of first scan in flat text file. The results include: Batch Tree, Tag List, and Equipment Tree. This parameter is designed primarily for troubleshooting.
/ps=x Required	The **/ps** parameter specifies the point source for the interface. x is not case sensitive and can be any multiple character string. For example, **/ps=P** and **/ps=p** are equivalent. The point source that is assigned with the **/ps** parameter corresponds to the PointSource attribute of individual PI Points. The interface will attempt to load only those PI points

Parameter	Description
	with the appropriate point source.
/PtSec=<string> Optional	The **/PtSec** parameter allows to specify the PIPoint Access Security rights. These rights are assigned only to interface generated tags during point creation. This string has different forms. If PIPoints are created on a PI Server 3.4.375.99 or earlier, it will have an owner, group, world format. Example: /ptsec="**o:rw g:r w:r**" If PIPoints are created on a PI Server 3.4.380.36 or later, it must specify an Access Control List (ACL) Example: /ptsec="piadmin: A(r,w) | PIEngineers: A(r)"
/restef Optional **AF Only**	(R)eferenced (E)lement (S)ecurity (T)o (E)vent(F)rame) command line parameters enables an event frame with references to inherit security settings from its Primary Reference Element.
/ras=<start, stop> Optional	The **/ras** Report As Step switch allows to use the "Report" event to create Phase Steps under active Phase States. The Phase Step name and start/stop events are obtained from the "Descript" column. Note: if the Phase Step left open, it is going to be closed by the end of the parent operation, and not by the end of parent phase or phase state. Example: /ras="-STRT, -STOP" Event - Report: Descipt Column: TEST123-STRT-B. Triggers the start of the Phase Step "TEST123" under the currently active Phase State. Event - Report: Descipt Column: TEST123-STOP-B. Sets an end time on the Phase Step "TEST123", if it can be found and regardless of what is the active Phase State.

Parameter	Description
/ret=<date time> Optional	The **R**ecovery **E**nd **T**ime **/ret** parameter is used to set the target end time of the history data recovery process. The Recovery End Time is approximate and interface is going to recover all batches with start time before Recovery End Time even though its end time might be beyond Recovery End Time.The <datetime> should be provided in local interface node time format. Note: This command must be used in conjunction with the optional switch: Recovery Start Time **/rst=<datetime>** Illustration:  Given Recovery Start – End timeframe, the interface is going to recover batches: B, C, D, E. Batches A and F are going to be ignored. Examples: **/ret="29-sep-2005 7:12:01 pm"** **/ret="07/20/2008 15:43:12"**
/Retry=<seconds> Optional Default: 60 seconds	The **/Retry** switch specifies the retry time delay, in seconds, for retrying a failed SDK attempt to write data to PI Server. The default retry delay is set to 60 seconds.
/RetryTO=<seconds> Optional Default: 0 seconds	The **Retry T**ime**O**ut **/retryTO** switch specifies the timeout, in seconds, for retrying a failed SDK attempt to write data to PI. The default timeout is set to 0 seconds (infinity). **Note:** To prevent data loses, it is recommended **NOT** to use the retry timeout switch.

Parameter	Description
/rst=<datetime> Optional	The **R**ecovery **S**tart **T**ime **/rst** parameter is used to set the target start time of the history data recovery process. The Recovery Start Time is approximate and an interface is going to recover all batches which start time after Recovery Start Time. In the boundary case when the batch start time is before Recovery Start Time and the batch end time is after Recovery Start Time, the interface is going to perform recovery for such batches as well. The <datetime> should be provided in local interface node time format. Illustration: Recovery Start / Recovery End ----------|----------|----------→ time line [--A--] [-----B-----] [-------------C-----------] [---D---] [---------E-------] [---F---* Given Recovery Start – End timeframe, the interface is going to recover batches: B, C, D, E. Batches A and F are going to be ignored. Examples: **/rst="29-sep-2003 7:12:01 pm"** **/rst="05/21/2007 15:43:12"**
/Scan=<seconds> Optional Default: **60** seconds	The **/Scan** parameter defines the time period between Interface scans in terms of seconds. Example: **/Scan=30** If the scanning frequency is set to 30 seconds, the Interface will attempt to query and process data every 30 seconds. Although, scan may be skipped if an abundance of data is processed.
/SingleRun Optional	This parameter forces the interface to perform only one scan and then stop.
/smp="PI Module Path" **or** **/smp="PI AF Element Path"** Optional	The **/smp** switch designates an alternate PI Module path, or PI AF element path if the PI AF database is used, to start looking for a particular Equipment hierarchy. If this option is not specified (i.e. the default) is to begin at the root level A path must be specified. This path is of the syntax: \\<RootModule>\<SubModule>\<…> e.g. \\MyEnterprise\MyPlant\
/SwapTime=<seconds> Optional	Defines the amount of time in seconds that the current primary interface must be unavailable before failover occurs. Default: 300 seconds. Example: /swaptime=240

Parameter	Description
/tbid Optional	Truncate BatchID. This parameter should be used in conjunction with **/bidm** parameter. When this parameter is enabled, all incoming events BatchID will be truncated according to the mask defined in **/bidm** switch. PIBatch, PIUnitBatch BatchID property will contain truncated BatchID. Tag and Property templates using placeholder [batchid] will replace it with truncated BatchID.
/tbse Optional	The **/tbse** True Batch Start End switch informs the interface to use actual top level recipe start/end events for creating the PI Batch objects. The original (default) behavior of the interface is to use the batch load/unload events. The new functionality (/tbse) is supported for batches with S88 recipe types: Procedure, UnitProcedure, Operation, and Phase.
/uidlist=<list> Optional	This parameter allows to recover specific Manufacturing Orders. If the switch is specified, on recovery completion the interface stops. The value should be defined as the list of the Manufacturing Order uniqueid's, which can be obtained from: **PASX.ManufacturingOrder.EntityKey** column Note: This parameter overrides the recovery **/rst** and **/ret** switches. Example 1: /uidlist=5008424880 Example 2: /uidlist=5010350293,5011438395

Sample PIFTBInt.bat File

The following is an example file:

```
REM=============================================================================
REM
REM PIFTBInt.bat
REM
REM Sample startup file for the PI Interface for FactoryTalk Batch
REM
REM=============================================================================
REM
REM Sample command line
REM
    PIFTBInt.exe ^
       /smp="\\Plant1" ^
       /host=XXXXXX:5450 ^
       /id=1 ^
       /ps=FTBINT ^
       /retry=30 ^
       /cachetime=0.1 ^
       /abto=30 ^
       /scan=45
REM
REM end of PIFTBInt.bat
```

Initialization File Parameters

The **Initialization** file: `PIFTBInt<serviceid>.ini` is used to specify the interface configurations, such as data sources, translations, product template, equipment template, tag templates and property templates. Note, most of the command line parameters can be defined in INI file. For example consider Recovery Start parameter **`/rst`** and **`/merge`** parameter.

The command line syntax:

`/rst="12/05/2008 12:05:23" /merge`

Equivalent Initialization file defined parameters:

`rst=12/05/2008 12:05:23`

`merge = true`

Note: In the initialization file each parameter should be defined on separate line. There should be only one equal (=) sign per line. Parameters can be disabled by specifying two forward slashes (**//**)

`//rst=12/05/2008 12:05:23`

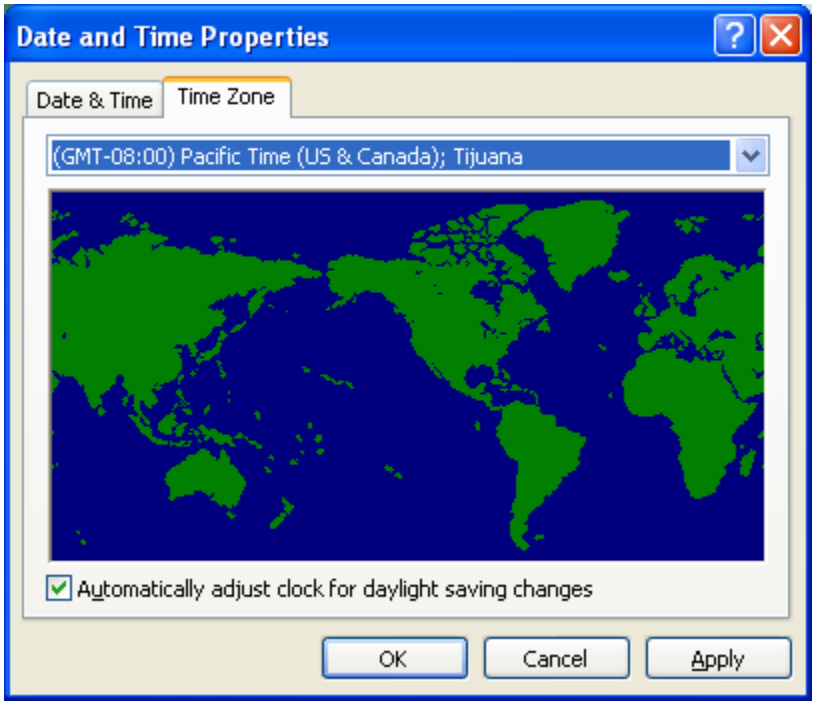
`//merge = true`

In this case, **`rst`** and **`merge`** parameters are disabled, therefore they are considered to be undefined.

The initialization can contain any free text. The only lines that will be loaded by the interface are lines with embedded equal sign and their continuation lines, if any. Sample INI files are included with the interface installation.

Chapter 9. Interface Node Clock

Make sure that the time and time zone settings on the computer are correct. To confirm, run the Date/Time applet located in the Windows Control Panel. If the locale where the interface node resides observes Daylight Saving Time, check the box marked "Automatically adjust clock for daylight saving changes". For example,



In addition, make sure that the TZ environment variable is not defined. All of the currently defined environment variables can be viewed by opening a Command Prompt window and typing `set`. That is,

```
C:> set
```

Confirm that TZ is not in the resulting list. If it is, run the System applet of the Control Panel, click the Environment Variable button under the Advanced Tab, and remove TZ from the list of environment variables.

Chapter 10. Security

The PI Firewall Database and the PI Proxy Database must be configured so that the interface is allowed to write data to the PI Server. See "Modifying the Firewall Database" and "Modifying the Proxy Database" in the PI Server manuals.

Note that the Trust Database, which is maintained by the Base Subsystem, replaces the Proxy Database used prior to PI version 3.3. The Trust Database maintains all the functionality of the proxy mechanism while being more secure.

See "Trust Login Security" in the chapter "Managing Security" of the *PI Server System Management Guide*.

If the interface cannot write data to the PI Server because it has insufficient privileges, a `-10401` error will be reported in the `pipc.log` file. If the interface cannot send data to a PI2 Serve, it writes a `-999` error. See the section Appendix A: Error and Informational Messages for additional information on error messaging.

PI Server v3.3 and Higher

Security configuration using piconfig

For PI Server v3.3 and higher, the following example demonstrates how to edit the PI Trust table:

```
C:\PI\adm> piconfig
@table pitrust
@mode create
@istr Trust,IPAddr,NetMask,PIUser
a_trust_name,192.168.100.11,255.255.255.255,piadmin
@quit
```

For the above,

`Trust`: An arbitrary name for the trust table entry; in the above example,

```
a_trust_name
```

`IPAddr`: the IP Address of the computer running the Interface; in the above example,

```
192.168.100.11
```

`NetMask`: the network mask; `255.255.255.255` specifies an exact match with `IPAddr`

`PIUser`: the PI user the Interface to be entrusted as; `piadmin` is usually an appropriate user

Security Configuring using Trust Editor

The Trust Editor plug-in for PI System Management Tools 3.x may also be used to edit the PI Trust table.

See the PI System Management chapter in the PI Server manual for more details on security configuration.

PI Server v3.2

For PI Server v3.2, the following example demonstrates how to edit the PI Proxy table:

```
C:\PI\adm> piconfig
@table pi_gen,piproxy
@mode create
@istr host,proxyaccount
piapimachine,piadmin
@quit
```

In place of `piapimachine`, put the name of the PI Interface node *as it is seen by PI Server*.

Chapter 11. Starting and Stopping the Interface

This section describes starting and stopping the interface once it has been installed as a service.

Starting Interface as a Service

If the interface was installed a service, it can be started from PI ICU, the services control panel or with the command:

```
PIFTBInt.exe -start
```

To start the interface service with PI Event Frames Interface Manager, use the "Start Interface Service" button on the Service Tab.

A message will inform you of the status of the interface service. Even if the message indicates that the service has started successfully, double check through the Services control panel applet. Services may terminate immediately after startup for a variety of reasons, and one typical reason is that the service is not able to find the command-line parameters in the associated `.bat` or initialization **`.ini`** file. Verify that the root name of the `.bat` file, `.ini` file and the `.exe` file are the same, and that the `.bat` file, `.ini` file and the `.exe` file are in the same directory. Further troubleshooting of services might require consulting the `pipc.log` file, Windows Event Viewer, or other sources of log messages. See the section Appendix A: Error and Informational Messages for additional information.

Stopping the Interface Running as a Service

If the interface was installed a service, it can be stopped at any time from PI Event Frames Interface Manager, the services control panel or with the command:

```
PIFTBInt.exe -stop
```

The service can be removed by:

```
PIFTBInt.exe -remove
```

To stop the interface service with PI Event Frames Interface Manager, use the "Stop Interface Service" button on the Service Tab

Chapter 12. Failover

Multiple interfaces can be configured to run in failover mode. Failover requires that the interfaces are configured identically, with identical BAT file, and identical INI file. Failover requires the configuration of three additional parameters:

Parameter Name	Value Type	Description
/FailOverID	<string>	The unique ID of the particular interface instance. The failover ID must be unique amongst the interfaces configured in failover mode
/FailOverTag	<PIPoint Name>	This PI Point is used to coordinate among participating interface instances which interface instance is primary.
/SwapTime	<time in seconds>	The amount of time that the current primary interface must be unavailable before failover occurs

When the interfaces are configured for failover, the current primary interface writes events to the failover tag. Each event has a timestamp of the current time and a value which has the format:

Failoverid | latest processed timestamp in UTC

Example:

interface1 | 1325376000

Each interface instance that is configured with this failover tag reads the current value of the failover tag. If the latest-processed timestamp occurred within the swap time, the current primary interface continues processing the data. If the latest-processed time stamp occurred prior to the swap time, a backup interface instance assumes the primary role. The new primary interface verifies the data and events in the cache time prior to the latest-processed timestamp, then resumes processing current data in real time.

Example:

Interface instance BAT file #1:

```
PIFTBInt.exe ^
/swaptime=30 ^
/failovertag="FTBInt1_FailoverTag" ^
/failoverid="intf1" ^
/rti ^
/inifile="C:\FTBInt\Test.ini" ^
```

```
/PS=EV ^
/ID=1 ^
/host=localhost2010 ^
/scan=10 ^
/cachetime=0.5 ^
/abto=10 ^
/db=1
```

Interface instance BAT file #2:

```
PIFTBInt.exe ^
/swaptime=30 ^
/failovertag="FTBInt1_FailoverTag" ^
/failoverid="intf2" ^
/rti ^
/inifile="C:\FTBInt\Test.ini" ^
/PS=EV ^
/ID=1 ^
/host=localhost2010 ^
/scan=10 ^
/cachetime=0.5 ^
/abto=10 ^
/db=1
```

Appendix A. Error and Informational Messages

A string `NameID` is pre-pended to error messages written to the message log. `Name` is a non-configurable identifier that is no longer than 9 characters. `ID` is a configurable identifier that is no longer than 9 characters and is specified using the **`/id`** flag on the startup command line.

Message Logs

The messages are logged in the local node log file *PIHOME\dat\pipc.log* **.**

Messages are written to log files at the following events:

- When the Interface starts many informational messages are written to the log. These include the version of the Interface, the version of PI SDK, the version of the PI Server, and the command-line parameters used.
- As the Interface processes batch-related data, messages are sent to the log if there are any problems with data retrieval from the data source or data processing to the PI Server.
- If the **`/db`** is used on the command line, then various informational messages are written to the log file.

Messages

The Batch interface logs all module, unit, alias, and point creation attempts for system management and auditing purposes. In addition, there are various debug level messages which may be logged using the **`/db=<level>`** switch in the interface startup file. See the section on Interface Operation for more detail on this switch.

Initialization or Startup Errors

Generally, these errors will stop the interface from starting up – it is normal behavior for the interface to exit since in many cases the proper startup state of the interface cannot be achieved (or determined) when these errors occur. Generally, speaking if an interface initialization error occurs, you should check to ensure that communications between the PI server and interface node are existent (since many of the initial parameters need to be synchronized – checked or created with or on the PI server).

"<source>: Memory Allocation Error, <error description>."

Errors, containing the message above, generally mean that the Interface node is out of memory. Release some memory by closing unused applications.

"<source>: COM Error: [error number]: <error description>."

Errors, containing the message above, are COM generated errors. These errors can occur on data retrieving from the data source as well as during processing of data to the PI Server. Refer to PI SDK reference manual for PI related COM errors to resolve such errors.

"<source> object = NULL" or "<source> pointer = NULL"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

"parse_argument_file: Error, Failed to open argument file: <argumentfile>"

This error means that the Interface failed to find the batch file associated with the specific Interface instance. Make sure that the batch file is consistent with the **serviceid** of the Interface. For example, on setup the service id is set as `serviceid 4`. In this case the batch file must be named `PIWPASXBatch4.bat`.

"parse_argfile_line: Error, Found open quote (\") without closing quote on command line...Terminating."

This error means that one of the command line parameters in the startup batch file has only one opening quote without matching closing quote. Check the batch file for missing quotes.

"read_ini_file: Error, unable to locate Initialization file: <filename>"

Verify that initialization file named <filename> exists in the Interface directory.

"read_ini_file: Error, unable to open Initialization file in READ MODE: <filename>"

Check the access properties of the initialization file named <filename>.

"read_startup_file: Error, unable to locate <startup file>: <filename>"

Verify that startup file named <filename> exists in the Interface directory.

"read_startup_file: Error, unable to open <startup file> in READ MODE: <filename>"

Check the access properties of the startup file named <filename>.

"write_startup_file: Error, failed to open <startup file> for writing : <filename>, Error: [errno=error number] :<error description>."

Check the access properties of the startup file named <filename>. Refer to error number and description for the actual error description.

"[REQUIRED PARAMETERS]: Development Error: No Batch Executive System defined. Please Contact OSIsoft technical support."

This is invalid build of the interface. Contact OSIsoft's technical support to request a valid build.

"[REQUIRED PARAMETERS]: Development Error: More than [1] Batch Executive System defined. Please Contact OSIsoft technical support."

The interface was build incorrectly; contact OSIsoft's technical support to request a valid build for required Batch Execution System.

"TemplateModuleList::Verify: Error, <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Equipment** module structure. Refer to error description for hints and check your input in initialization file.

"[REQUIRED PARAMETERS]: <error description>"

OR

"[MISSING REQUIRED COMMAND LINE PARAMETERS] : <error description>"

The errors containing message above generally mean that there are missing parameters in either command line or in initialization file required for interface startup. Please refer to error description to resolve the error.

"Main: Error, Failed to set Numerical Settings to : [language]"

The value provided for **`/ns`** switch in command line parameters is invalid, please check your input.

"check_SDK_version: Error: Too Many fields in PI SDK Version"

The interface failed to identify the PI SDK version number. Please consult with OSIsoft technical support to resolve this error.

"check_SDK_version: Error, This is an Old PI SDK Version, Please upgrade to <minimum SDK version> or higher."

The PI SDK version installed on the interface node is lower than the minimum required by the interface version of the PI SDK. Please download and install new version of PI SDK.

"set_PISDK_GUID: Error, The Interface failed to identify itself to the PI Server, appID = NULL. Terminating."

The interface failed to broadcast its Global Unique ID to the PI server. Please contact OSIsoft technical support to resolve this error.

"OpenPIConnection: Error, PI Server <collective name\PI server name> is a SECONDARY Server. The interface is designed to run only against PRIMARY PI Server. Terminating."

The current version of the interface is designed to run only against primary server if used in the Collective configuration. Change the **`/host`** switch value and restart the interface.

"netsafe::FindCreateMonitorTags: ERROR, Failed to Add <object description>"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

"StartHealthMonitor: Error, Failed to start health monitor thread. [error number]:<error description>"

This is windows related error, check error description.

"ReadCommandFile: ERROR, Unable to read Command file: <filename>, REASON: NO reading privileges"

Check the access properties of the command file named <filename>.

"ReadCommandFile: ERROR, Unable to reset Command file: <filename>, REASON: NO writing privileges"

Check the access properties of the command file named <filename>.

"mCOMThreadProc: ThreadID: [thread ID]: Error, Unable to retrieve passed arguments... Terminating"

This error indicates that the interface node might be out of memory. Release some memory by closing unused applications and restart the interface.

"The source IP address <server name> is not valid, <error description>"

The errors containing message above generally mean that the **`/host=<server name>`** switch value is invalid. Please refer to error description and correct your input in command line parameters.

"SourceList::AddUpdate: Error <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **source[#]** properties. Refer to error description for hints and check your input in initialization file.

"TemplatePropertyList::Verify: Error, <error description>"

or

"TemplatePropertyList::Add: Error, <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Property[#]** template value structure. Refer to error description for hints and check your input in initialization file.

"TemplateTagList::Verify: Error, <error description>"

or

"TemplateTagList::AddUpdate: Error <error description>"

The errors containing message above mean that there is an incorrect data provided while defining **Tag[#]** template properties. Refer to error description for hints and check your input in initialization file.

Runtime Errors

Generally, Batch interface errors are triggered by some action that the interface takes while interacting with the PI Server or reading data from the data source. Therefore, most (if not all) errors will contain a variable portion of the message which is returned from either the PI Server or the underlying PI SDK layers. PI server specific portions of messages will generally contain a negative five-digit number (e.g. –10401 or –15001). These numbers are often followed by a description. However, these error numbers can also be looked up using the following command line commands:

```
pidiag –e <error number>
```

or:

```
pilogsrv –e <error number>
```

PI SDK numbers are generally eight-digit hexadecimal numbers (e.g. 0x000403a0). Again specific descriptions for the error are generally appended to the error message, but can also be obtained by using the "Error Lookup" function in the AboutPI SDK.exe application installed when the PI SDK is installed.

"<source>: Memory Allocation Error, <error description>."

Errors, containing the message above, generally mean that the Interface node is out of memory. Release some memory by closing unused applications.

"<source>: COM Error: [error number] : <error description>."

Errors, containing the message above, are COM generated errors. These errors can occur on data retrieving from the data source as well as during processing of data to the PI Server. Refer to PI SDK reference manual for PI related COM errors to resolve such errors.

"<source>: Critical Error, <error description>."

"<source> object = NULL" or "<source> pointer = NULL"

Errors, containing the messages above, are memory allocation related errors. Generally mean that the Interface node is out of memory. Release some memory by closing unused applications and restart the interface.

System Errors and PI Errors

System errors are associated with positive error numbers. Errors related to PI are associated with negative error numbers.

Error Descriptions

On Windows, descriptions of system and PI errors can be obtained with the `pidiag` utility:

```
\PI\adm\pidiag –e error_number
```

Appendix B. Technical Support and Resources

You can read complete information about technical support options, and access all of the following resources at the OSIsoft Technical Support Web site:

http://techsupport.osisoft.com (http://techsupport.osisoft.com)

Before You Call or Write for Help

When you contact OSIsoft 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 log file(s) at that time

Help Desk and Telephone Support

You can contact OSIsoft Technical Support 24 hours a day. Use the numbers in the table below to find the most appropriate number for your area. Dialing any of these numbers will route your call into our global support queue to be answered by engineers stationed around the world.

Office Location	Access Number	Local Language Options
San Leandro, CA, USA	1 510 297 5828	English
Philadelphia, PA, USA	1 215 606 0705	English
Johnson City, TN, USA	1 423 610 3800	English
Montreal, QC, Canada	1 514 493 0663	English, French
Sao Paulo, Brazil	55 11 3053 5040	English, Portuguese
Frankfurt, Germany	49 6047 989 333	English, German
Manama, Bahrain	973 1758 4429	English, Arabic
Singapore	65 6391 1811 86 021 2327 8686	English, Mandarin Mandarin
Perth, WA, Australia	61 8 9282 9220	English

Support may be provided in languages other than English in certain centers (listed above) based on availability of attendants. If you select a local language option, we will make best efforts to connect you with an available Technical Support Engineer (TSE) with that language skill. If no local language TSE is available to assist you, you will be routed to the first available attendant.

If all available TSEs are busy assisting other customers when you call, you will be prompted to remain on the line to wait for the next available TSE or else leave a voicemail message. If you choose to leave a message, you will not lose your place in the queue. Your voicemail will be treated as a regular phone call and will be directed to the first TSE who becomes available.

If you are calling about an ongoing case, be sure to reference your case number when you call so we can connect you to the engineer currently assigned to your case. If that engineer is not available, another engineer will attempt to assist you.

Search Support

From the OSIsoft Technical Support Web site, click *Search Support*.

Quickly and easily search the OSIsoft Technical Support Web site's Support Solutions, Documentation, and Support Bulletins using the advanced MS SharePoint search engine.

Email-based Technical Support

techsupport@osisoft.com

When contacting OSIsoft Technical Support by email, it is helpful to send the following information:

- Description of issue: Short description of issue, symptoms, informational or error messages, history of issue
- Log files: See the product documentation for information on obtaining logs pertinent to the situation.

Online Technical Support

From the OSIsoft Technical Support Web site, click *Contact us* > *My Support* > *My Calls*.

Using OSIsoft's Online Technical Support, you can:

Enter a new call directly into OSIsoft's database (monitored 24 hours a day)

View or edit existing OSIsoft calls that you entered

View any of the calls entered by your organization or site, if enabled

See your licensed software and dates of your Service Reliance Program agreements

Remote Access

From the OSIsoft Technical Support Web site, click *Contact Us* > *Remote Support Options*.

OSIsoft Support Engineers may remotely access your server in order to provide hands-on troubleshooting and assistance. See the Remote Access page for details on the various methods you can use.

On-site Service

From the OSIsoft Technical Support Web site, click *Contact Us* > *On-site Field Service Visit*.

OSIsoft provides on-site service for a fee. Visit our On-site Field Service Visit page for more information.

Knowledge Center

From the OSIsoft Technical Support Web site, click *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 on 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 checks, daylight savings time configuration, PI Server security, PI System sizing and configuration, PI trusts for Interface Nodes, and more.

Upgrades

From the OSIsoft Technical Support Web site, click *Contact Us* > *Obtaining Upgrades*.

You are eligible to download or order any available version of a product for which you have an active Service Reliance Program (SRP), formerly known as Tech Support Agreement (TSA). To verify or change your SRP status, contact your Sales Representative or *Technical Support* (http://techsupport.osisoft.com/) for assistance.

OSIsoft Virtual Campus (vCampus)

The OSIsoft Virtual Campus (vCampus) Web site offers a community-oriented program that focuses on PI System development and integration. The Web site's annual online subscriptions provide customers with software downloads, resources

that include a personal development PI System, online library, technical webinars, online training, and community-oriented features such as blogs and discussion forums.

OSIsoft vCampus is intended to facilitate and encourage communication around PI programming and integration between OSIsoft partners, customers and employees. See the OSIsoft vCampus Web site, *http://vCampus.osisoft.com* (http://vCampus.osisoft.com) or contact the OSIsoft vCampus team at vCampus@osisoft.com for more information.