LISTEN. THINK. SOLVE.®

INTEGRATED PRODUCTION & PERFORMANCE SUITE

Data Management

FactoryTalk® Historian SE

FACTORYTALK HISTORIAN TO HISTORIAN INTERFACE USER GUIDE

PUBLICATION H2H-UM001A-EN-E–July 2012

Allen-Bradley • Rockwell Software

Rockwell Automation

Table of Contents

Chapter 1. Introduction

The Rockwell FactoryTalk Historian to Historian interface copies tag data from one Historian Server to another. Data is moved in one direction, meaning data is copied from the source to the receiving Historian Server (also referred to as target Historian Server). The interface must run on a Windows Intel-based operating system.

Note: The Rockwell FactoryTalk Historian to Historian interface functions and is configured the same as the standard Rockwell Automation PItoPI interface, with the exception that this interface can only connect to FT Historians.

Interface tags are created on the receiving Historian Server. Each interface tag is configured to receive data for a unique source tag. Tags receive either archive or exception data updates from the source tag. Exception data is data that has not yet been subjected to compression. The type of data collection, exception or archive, is configured through scan class assignment. By default, all tags belonging to the first scan class receive exception data. Tags assigned to any other defined scan class receive archive data.

The interface supports history recovery. History recovery enables users to recover data for time periods when the interface was not running or otherwise unable to collect data. The history recovery period is configurable; the default is 8 hours. Users have the option of performing time-range specific history recovery by specifying a start and end time. In this configuration the interface collects data for the specified time period then exits.

Both source Historian Server-level failover and UniInt Phase 2 interface level failover are supported. When running in source Historian Server-level failover mode, the interface obtains data from one of two available source Historian Servers. The source Historian Servers must have identical tag definitions and data streams for each interface source tag. This requirement ensures the interface will obtain the same data regardless of which source server is active. When running in UniInt Failover mode, two copies of the interface are connected to the source Historian Server at the same time. When the primary interface stops collecting data, the backup interface assumes the primary role and continues data collection. Source Historian Server-level failover and UniInt Phase 2 interface level failover modes can be run simultaneously. Failover maximizes interface data availability on the receiving Historian Server(s).

Interface Limitations

The FactoryTalk Historian to Historian interface is not a true data replication tool. It does not synchronize Historian Server data or perform data validation. It simply provides a method for copying data from one Historian Server to another in an incremental, time forward manner. There is no guarantee that an exact archive data match will exist between the source and receiving Historian Servers. If the goal is to achieve data matching (replication) Rockwell

Automation recommends using n-way buffering which is supported with PI API v1.6.x and later. Please see the PI API installation manual for details.

The Historian Archive subsystem may temporarily queue data in memory prior to it being committed to disk. This can lead to data gaps when using Historian to Historian for real-time data collection with history recovery enabled. To avoid data gaps the recommended configuration is to run in history recovery only mode without snapshot updates. Note that this means current real-time data from the source Historian Server will not be available on the target Historian Server.

The interface is a PI API based application. It does not currently support tag annotations, which are only available through the PI SDK. This means it cannot be used to copy Batch Database data between Historian Servers.

The interface is a single threaded process. This design increases performance dependencies on the responsiveness of the source and receiving Historian Servers and dependencies on network quality.

It is highly recommended that users use tools such as Rockwell Automation’s Performance Monitor interface and Historian Ping interface to monitor these interface dependencies. These interfaces are distributed by default with the latest Historian Server setup kit. This information will be invaluable for troubleshooting Historian to Historian interface issues if they should arise. Using these tools to monitor system health is also part of Rockwell Automation’s Best Practices Recommendations for FactoryTalk Historian System Mangers.

Interface Requirements

PItoPI requires the following software versions:

- Windows Intel-based operating system
- PI API version 1.6.1.5 or greater, which is distributed with the PI SDK installation kit.
- TCP/IP connections are needed to both receiving and source Historian Servers. This interface also operates using RAS to connect over dial-up or ISDN connections. Dial-up connections of 9600 baud have been used successfully.
- To configure Phase 2 failover using the ICU you must use ICU 1.4.6.0 or later.

Note: The value of [PIHOME] variable for the 32-bit interface will depend on whether the interface is being installed on a 32-bit operating system (`C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC`) or a 64-bit operating system (`C:\Program Files (x86)\PIPC`).

The value of [PIHOME64] variable for a 64-bit interface will be C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC on the 64-bit Operating system.

In this documentation [PIHOME] will be used to represent the value for either [PIHOME] or [PIHOME64]. The value of [PIHOME] is the directory which is the common location for Historian client applications.

Reference Manuals

Rockwell Automation

- *Historian Server manuals*
- *PI API Installation manual*
- *UniInt Interface User Manual*
- *Historian Interface Status*

Supported Operating Systems

Platforms		32-bit application	64-bit application
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 7	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

The interface is designed to run on the above mentioned Microsoft Windows operating systems and their associated service packs.

Please contact Rockwell Automation Technical Support for more information.

Supported Features

Feature	Support
Interface Part Number	Historian-IN-RW-FTPI-NTI
* Auto Creates Historian Points	APS Connector
Point Builder Utility	No
ICU Control	Yes
Historian Point Types	Historian 3: Float16 / Float32 / Float64 / Int16 / Int32 / Digital / String / Blob Historian 2: R / I / D
Sub-second Timestamps	Yes
Sub-second Scan Classes	Yes
Automatically Incorporates Historian Point Attribute Changes	Yes
Exception Reporting	Yes
Outputs from Historian	No

Feature	Support
Inputs to Historian: Scan-based / Unsolicited / Event Tags	Scan-based Only
* Supports Questionable Bit	No
Supports Multi-character PointSource	Yes
Maximum Point Count	Unlimited
* Uses PI SDK	No
PINet String Support	No
* Source of Timestamps	Source Historian Server
History Recovery	Yes
UniInt-based * Disconnected Startup * SetDeviceStatus	Yes No Yes
Failover	Source Historian Server-Level UniInt Phase 2 Interface Level (Warm)
* Vendor Software Required on Historian interface node / PINet Node	No
Vendor Software Required on Foreign Device	No
Vendor Hardware Required	No
Additional Historian Software Included with Interface	No
Device Point Types	Real, Integer and Digital.
Serial-Based Interface	No

** See available paragraphs below for further explanation.*

APS Connector

The PItoPI APS Connector (PItoPI_APS) is a specific module that communicates with the Receiving and Source Host FactoryTalk Historian SE's, gets tag attribute updates from the Source Host FactoryTalk Historian System, and locates new and deleted points on the Source Host FactoryTalk Historian System. The PItoPI APS Connector and its attendant routines are called with each synchronization scan.

Note: The PItoPI APS Connector contains a separate implementation of the procedure to identify the source point for an interface point. If an interface instance is registered with APS, confirm that the PItoPI APS Connector version implements the same procedure as the interface, which is required to ensure that the PItoPI APS Connector synchronizes with the same source point as the interface.

Supports Questionable Bit

The interface will copy questionable bit data for a give source point from one PI3 server to another. However the interface itself does not make this a configurable parameter.

Uses PI SDK

The PI SDK and the PI API are bundled together and must be installed on each interface node. This Interface does not specifically make PI SDK calls.

Maximum Point Count

While the interface does not have a hard-coded point count limit, there are performance dependencies that can effectively limit point count. The interface is a single-threaded process. This design exposes performance dependencies on Historian Server responsiveness and network quality. For example a high latency network connection will result in a significant decrease in data transfer rate over a network connection without high latency.

It is highly recommended that users use tools such as Rockwell Automation’s Performance Monitor interface and Historian Ping interface to monitor these interface dependencies. These interfaces are distributed by default with the latest Historian Server setup kit. This information will be invaluable for troubleshooting Historian to Historian interface issues if they should arise. Using these tools to monitor system health is also part of Rockwell Automation’s Best Practices Recommendations for FactoryTalk Historian System Mangers.

Source of Timestamps

The source Historian Server provides a timestamp for each data event. Timestamps are automatically adjusted to account for time zone differences if both source and receiving Historian Servers are Historian 3. Time zone adjustment is optional if either Historian Server is Historian 2.

The interface can also adjust timestamps for clock drift. Clock drift is the time offset between Historian Servers after accounting for time zone differences. An offset of 30 minutes or less is considered clock drift. Adjusting for clock drift means the time offset is added to the source timestamp adjusting it to receiving Historian Server time.

Timestamp adjustment is configured on a tag-by-tag basis through the `Location2` tag attribute. Note that all computers (interface nodes, source and receiving Historian Servers) must have the correct system time for their configured time zone.

History Recovery

History recovery enables users to recover archive data for time periods when the interface was not running or otherwise unable to collect data. History recovery is performed on startup, after restoring a lost Historian Server connection and after a disruption in exception data collection. In addition, when a new point is added to the interface, history recovery is performed on that point. The history recovery period is configurable. The default is to recover data for the previous 8 hours. History recovery is disabled by setting the time period to 0.

Time range-specific history recovery can be performed by passing a start and end time. When run in this configuration the interface collects data for the specified time range then exits. It should be noted that the start and end time will be relative to the node where the interface runs. This is important if the source Historian Server is in a different time zone than the machine where the interface is running.

UniInt-based

UniInt stands for Universal Interface. UniInt is not a separate product or file; it is an Rockwell Automation-developed template used by developers, and is integrated into many interfaces, including this interface. The purpose of UniInt is to keep a consistent feature set and behavior across as many of Rockwell Automation’s interfaces as possible. It also allows for the very rapid development of new interfaces. In any UniInt-based interface, the interface uses some of the UniInt-supplied configuration parameters and some interface-specific parameters. UniInt is constantly being upgraded with new options and features.

The *UniInt Interface User Manual* is a supplement to this manual.

SetDeviceStatus

The Historian to Historian Interface is built with UniInt 4.4.4.0. New functionality has been added to support health tags. The Health tag with the point attribute `ExDesc = [UI_DEVSTAT]` represents the status of the source device. The following events can be written into this tag:

"1 | Starting" - the interface is starting.

- "Good" - the interface is properly communicating and reading data from the server.
- The following event represents a failure to communicate with the server:
  - "3 | 1 device(s) in error | Network communication error to source Historian Server"
  - "3 | 1 device(s) in error | Unable to get archive data from source Historian Server"
  - "3 | 1 device(s) in error | Unable to get snapshot data from source Historian Server"
  - "3 | 1 device(s) in error | Unable to write data to receiving Historian Server"
  - "3 | 1 device(s) in error | Unable to obtain current data with source Historian Server failover enabled."
- "4 | Intf Shutdown" - the interface is stopped.

Refer to the *UniInt Interface User Manual* for more information on how to configure health points.

Failover

Source Historian Server-Level Failover Support

Source Historian Server-level failover maximizes interface data availability on the receiving Historian Server(s). It enables the interface to obtain data from one of two source Historian Servers. Each source server must have identical tag definitions and data streams for interface source points. Source Historian Server failover is not supported for Historian 2.

The interface will initiate failover if the active source data becomes stale or is not available due to network issues. The Historian Interface Status utility is required for each source Historian Server to monitor data quality. The interface uses the utility status tag output to verify source data is current and has not become stale.

UniInt Interface Failover Support

UniInt Phase 2 Failover provides support for cold, warm, or hot failover configurations. The Phase 2 hot failover results in a no data loss solution for bi-directional data transfer between the Historian Server and the Data Source given a single point of failure in the system architecture similar to Phase 1. However, in warm and cold failover configurations, you can expect a small period of data loss during a single point of failure transition. This failover solution requires that two copies of the interface be installed on different interface nodes collecting data simultaneously from a single data source. Phase 2 Failover requires each interface have access to a shared data file. Failover operation is automatic and operates with no user interaction. Each interface participating in failover has the ability to monitor and determine liveliness and failover status. To assist in administering system operations, the ability to manually trigger failover to a desired interface is also supported by the failover scheme.

The failover scheme is described in detail in the *UniInt Interface User Manual*, which is a supplement to this manual. Details for configuring this Interface to use failover are described in the UniInt Failover Configuration section of this manual.

Device Point Types

Real, Integer and Digital.

Diagram of Hardware Connection

The following diagrams illustrate the three basic hardware configurations for PItoPI: the interface runs on the source Historian Server, the interface runs on the receiving Historian Server or the interface runs on a separate interface node. The interface must run on a Windows Intel-based operating system.

Figure 1: Configuration 1 PItoPI runs on the source Historian Server "pushing" data to the receiving Historian Server.

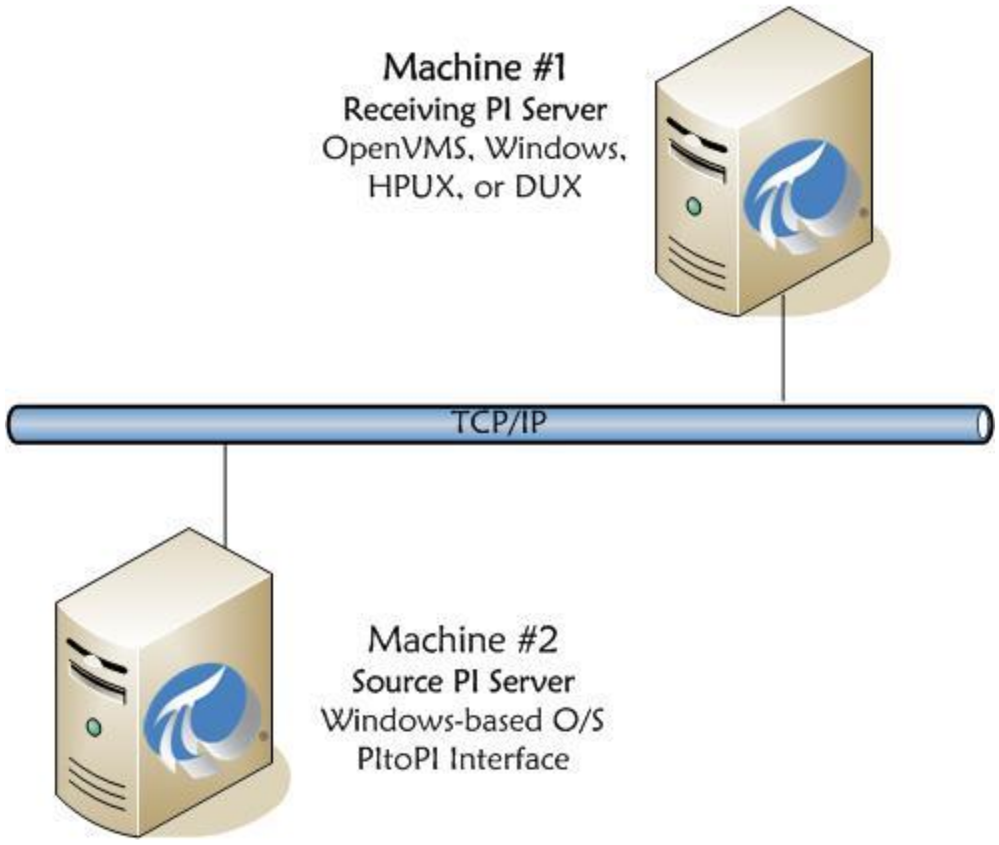


Figure 2: Configuration 2 PItoPI runs on the receiving Historian Server "pulling" data from the source Historian Server.

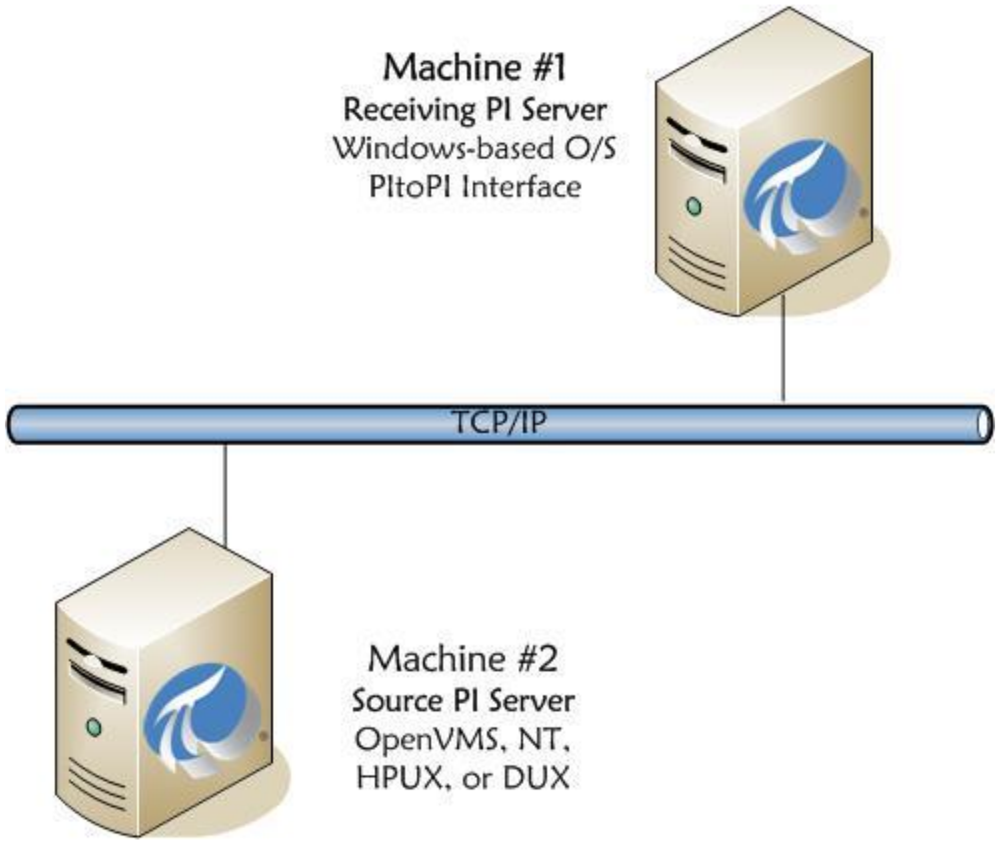
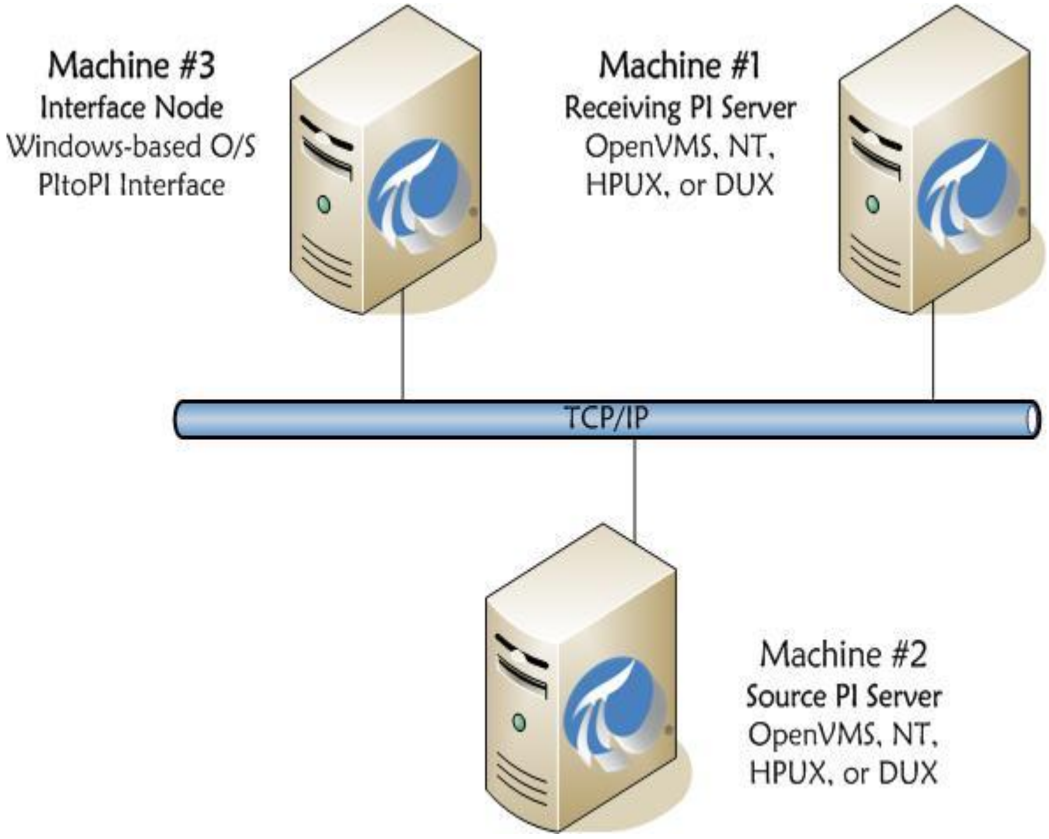


Figure 3: Configuration 3 PItoPI runs on a separate interface node.



Chapter 2. Principles of Operation

The FactoryTalk Historian to Historian interface copies tag data from one Historian Server to another. It consists of a single executable and startup configuration file. No specific tag limit exists for the interface. However, data throughput (events/second) can be limited by network quality and/or system resources on the source and receiving Historian Servers. Users can configure the interface to copy exception (snapshot) or archive data. Although it is possible for one copy of the interface to do both, it is highly recommended that separate copies are used for this purpose to ensure a high level of performance.

Interface Startup

On startup, the interface reads its configuration file for site-specific operating parameters. This file contains required and optional configuration information, such as source and receiving Historian Server, data update rates, and history recovery settings. Next the interface establishes a connection to each Historian Server and initializes its tag lists.

Each client connection to a Historian Server is associated with a specific PI user. This PI user determines what permissions are granted to the client. The source Historian Server must grant the interface permission to read tag attributes and data for interface source tags. On the receiving Historian Server the interface requires read access for tag attributes and read and write data access for its tag list. See the section Tag and Node Security for additional information.

As the interface initializes its tag lists, tags are grouped internally by scan class assignment. Each interface tag must be configured for a unique source tag. If an invalid tag configuration is detected, the tag is either rejected or added to the tag list and marked in error. In either case a message will be logged with specific information about the error. The interface outputs messages to `pipc.log`, which can be found in the local `\pipc\dat` directory.

After the interface is finished building its tag lists it calculates the time offset between Historian Servers. The offset is updated every 2 minutes and used for data timestamp adjustments, if configured, and to timestamp interface status events. It is required that each participating node has the correct system time for its configured time zone. At this point the interface is ready to begin data collection.

How FactoryTalk Historian to Historian Finds Source Points

When the FactoryTalk Historian to Historian Interface loads a point, the interface must identify the source point from which data will be collected. In other parts of this manual, this process is referred to as *mapping* the interface point to a source point.

The interface uses four receiving point attributes as possible links to the source point: `InstrumentTag`, `ExDesc`, `UserInt1`, and `Tag`. However, the FactoryTalk Historian to Historian Interface **`/tn`**, **`/tnex`**, and **`/ptid`** parameters exclude one or more of these attributes from mapping to the source point. For most interface instances, none of these parameters are used. The following table summarizes the effect of these parameters on the search for either source point tag name or point ID. The actual implementation of the search is described below the table.

Interface parameters			**Search order for attribute containing source point tag name or ID**
/tn	**/tnex**	**/ptid**	
No	No	No	1. Non-empty InstrumentTag attribute is source point tag name. 2. ExDesc attribute that contains STAG= specifies source point tag. 3. UserInt1 attribute greater than 0 contains source point ID. (Point rejected if source Historian Server-level failover configured.) 4. Tag of the interface point is also source point tag name.
No	No	Yes	UserInt1 attribute contains source point ID. (Illegal configuration if source Historian Server -level failover is configured.)
No	Yes	No	1. ExDesc attribute that contains STAG= specifies source point tag. 2. Tag of the interface point is also source point tag name.
No	Yes	Yes	Tag of the interface point is also source point tag name.
Yes	N/A	N/A	

The interface performs the following steps to find the source point:

1. If no **`/tn`** parameter and no **`/tnex`** parameter and no **`/ptid`** parameter and the `InstrumentTag` attribute is not a zero-length string, the `InstrumentTag` value contains the source tag name and the search ends. Otherwise, proceed to step 2.

2. If no **`/tn`** parameter and no **`/ptid`** parameter and the `ExDesc` attribute begins with case-insensitive "STAG" followed by zero or more spaces followed by "=", the source tag name is extracted from the remainder of the `ExDesc` value (as described in the following paragraph) and the search ends. Otherwise, proceed to step 3.

   To extract the source tag name, the interface ignores any spaces following the "=". If the next character is not a double quotation mark ("), it is the first character of the source tag name which extends to the first comma or to the end of the `ExDesc` attribute. If the first non-space following the "=" is a double quotation mark, the source tag name begins with the following character and extends to the first double quotation mark or to the end of the `ExDesc` attribute. The interface extracts the source tag name from `ExDesc` and the search ends.

3. If no **`/tn`** parameter and no **`/tnex`** parameter and either the `UserInt1` attribute is greater than zero or the **`/ptid`** parameter is present, the `UserInt1` attribute is the

source point ID and the search ends. Otherwise, proceed to the step 4. If the search ends in this step and either the `UserInt1` attribute is not greater than zero or the interface is configured for source Historian Server-level failover, the interface rejects the point.

4. The source point tag name is the same as the receiving tag name.

The search for the source point ends with either a source tag name or source point ID. If the source tag name or point ID does not exist, the interface puts the point into an error state and logs a message.

Note: The PItoPI APS Connector contains a separate implementation of the search for source point tag name or ID. If an interface instance is registered with APS, confirm that the PItoPI APS Connector version implements the same search procedure as the interface, which is required to ensure that the PItoPI APS Connector synchronizes with the same source point as the interface. If the latest release of the PItoPI APS Connector is older than this version of the FactoryTalk Historian to Historian interface, compare the procedure described in this section with the “How PItoPI and PItoPI_APS Find Source Points” section in the *Historian to Historian TCP/IP Interface AutoPointSync Connector* user manual.

Data Collection

There are two modes of data collection, history recovery and scanning for updates. History recovery enables users to recover archive data for time periods when the interface was not running or otherwise unable to collect data. After performing history recovery, the interface begins scanning for updates. When the interface is scanning for updates each tag recei Historian to Historian TCP/IP Interface AutoPointSync Connector ves an incremental copy of either archive or exception data from its configured source tag.

A tag configured for exception data receives a copy of its source tag’s current value updates. A tag’s current value is also referred to as its snapshot value. A snapshot value is a data event that has passed the first level of data filtering, the exception test. Every Historian Server maintains a snapshot table that contains the current value for each tag in its database. When a new value passes exception filtering, the snapshot table is updated. The existing snapshot value is then subjected to the compression test. The compression test determines whether or not a snapshot value is archived. If a value passes compression filtering, it becomes part of the archive data collection.

Tags are configured for archive or exception data collection through scan class assignment. Scan classes are update rates that users define in the interface startup configuration file. When a scan is executed the interface performs an incremental copy of data for each tag in its scan list. By default, all tags assigned to the first scan class receive exception data. Tags assigned to any other scan class receive archive data.

Note: Interface performance is maximized when a separate copy of the interface is used for archive and exception data collection.

All data collected by the interface has already passed exception filtering on the source Historian Server. Any additional exception filtering can only lead to a data mismatch between Historian Servers. Additionally we recommend applying source tag compression deviation settings to interface tags. See the section Exception and Compression for additional information.

In general the interface can sustain higher data rates (events/second) when configured for exception data collection. Exception updates are queued in memory on the source Historian Server. This is not the case for archive updates. Only the most recently accessed archive data is stored in memory, specifically the archive data cache. When the interface requests archive data it is likely a certain percentage is read from disk. Disk reads require more computer resources than reading from memory. As the interface data rate increases, the source Historian Server will work harder if the interface is requesting archive versus exception data.

History Recovery

History recovery is executed after each of the following events:

- On interface startup.
- After restoring a lost Historian Server connection.
- Update overflow error recovery.
- For each new point added to the interface.

History recovery is configured through interface startup parameters. Users can specify the recovery period, time increments within the total recovery period, a pause time between increments, and the maximum number of archive events returned per data request. These parameters enable users to manage the performance impact on the source and receiving Historian Servers by throttling the data collection rate.

The starting point for history recovery is set on a tag-by-tag basis. If the current value for a tag is more recent than the starting point of the recovery period, history recovery will begin at the tag's current value. This prevents data overlap, streamlines data retrieval, and prevents out-of-order data. There is one exception: `Pt Created` is the value given to a tag when it is created. If the current value of an interface tag is `Pt Created`, the interface performs history recovery for the total recovery period.

History recovery is performed independently for each scan class. The interface periodically prints a message to indicate completion status. Every minute the interface calculates percent completion. A message will be printed for each 10% completion increment or every 5 minutes, whichever comes first.

If a scan class is configured for exception data collection, the interface transitions from archive to exception data on the last history recovery increment. As the interface cycles through its tag list, each tag is added to the exception update list after obtaining its last recovery increment. By default the interface breaks from history recovery to collect exception updates every 5 seconds. This time interval is configurable through the **`/rh_qcheck`** startup parameter. As it nears the end of its tag list, more tags are in the update list and the time to collect exception updates increases. This extra overhead can significantly increase the time for complete history recovery.

The default behavior is for history recovery data to bypass compression on the receiving Historian Server. Source data is written to the receiving archive in one of three modes; append, replace or no replace. This is configured on a tag-by-tag basis. If the **`/dc`** interface startup parameter is specified, this data is subjected to compression. Historian 2 and Historian 3.2 servers will always apply compression to interface data. In this case the data write mode is effectively disabled.

Time-range history recovery is configured through the interface startup configuration file. When enabled, the interface starts up, recovers archive data for the specified start and end time then exits. The times specified are relative to the machine where the interface runs. This is important if the source Historian Server is in a different time zone than the machine where the interface runs. Note that backfilling data may require additional server preparations on the receiving node. For example, there may not be enough space in the target (non-primary) archive for the recovery data, or non-primary archives may not have space allocated for newly created tags. See the *Historian Server System Management Guide* for information on backfilling data.

Exception Data Collection

Tags assigned to the first scan class receive exception data. An exception is a current value update (snapshot value). On startup each source tag is registered with the update manager on the source FactoryTalk Historian System. The update manager then collects snapshot updates for each registered tag. This data is queued in memory. When the interface executes a scan it requests and processes these events until the queue is empty. Data update latency is minimized by configuring a high frequency scan rate for the exception scan class. Exception and compression should be configured carefully for the interface to avoid data mismatches between Historian Servers. See the section Exception and Compression for additional information.

Maximizing Data Throughput

Interface performance is maximized when a separate copy of the interface is used for archive and exception data collection.

The number of events returned per update request to the source Historian Server is configurable through the interface startup file. The default is for up to 10,000 exception events returned per update request. This list of exceptions is given to the interface in time sequence, oldest to newest. The interface processes the list one event at a time. These events are temporarily queued by the interface. Only one update is queued per tag at any given time. Whenever a second event for any one tag is processed within the exception update list, the interface must first flush its queue, writing this data to the receiving server. This behavior is required for exception data filtering. As a result, the interface will make many more calls writing data to the receiving Historian Server than retrieving data from the source.

Note: Whenever possible, users should run the interface on the receiving Historian Server to minimize the effect of network latency on data throughput.

Network latency will have a significant effect on data throughput. Interface testing over a WAN connection with 200ms latency showed throughput was reduced by 2/3 when the interface ran on the source versus the receiving Historian Server. When run on the receiving Historian Server, the interface was able to sustain 1800 events/second. This was reduced to 600 events/second when the interface ran on the source Historian Server. Testing with a latency of <1 ms, throughput was not affected. The interface maintained 11,000 events/second when run on the source and receiving Historian Server.

Archive Data Collection

By default, the first scan class is used for exception data collection. Tags configured for any other scan class will receive archive data. However, exception data collection can be disabled entirely by specifying the **`/hronly`** interface startup switch. When specified, all scan classes update with archive data.

When a scan is executed, each tag receives an incremental copy of data. An archive update begins at the interface tag's current value. The source server returns all source tag data including its current snapshot value. Users have the option of including the current snapshot value with each scan update. This is configured through the `Location3` tag attribute. The snapshot value is not part of the archive data collection. Including the snapshot value can lead to a data mismatch between Historian Servers. In this case data compression must be enabled by specifying the **`/dc`** startup switch. If this is not done and a tag is configured to include the snapshot, a snapshot value will be archived on each scan. This will result in the interface tag having more archive values than its source.

The default behavior is for archive data collected by the interface to bypass compression on the receiving Historian Server if its version is PI 3.3 or later. Source data is written to the receiving archive in one of three modes; append, replace or no replace. This is configured on a tag by tag basis through the `Location5` tag attribute. If the **`/dc`** interface startup parameter is specified, this data is subjected to compression which effectively disables the archive write mode. Historian 2 and Historian 3.2 servers will always apply compression to interface data. In this case compression must be managed through interface tag configurations. See the section Exception and Compression for additional information.

Performance Considerations

Load Distribution

Archive data collection can have a significant impact on Historian Server performance. The source server will be providing the interface with outgoing data for its tag list in addition to the normal incoming data rate. This may significantly increase system resource usage. This performance impact can be minimized by distributing tags among multiple scan classes.

For example, a user configures the interface for 10,000 tags. Instead of assigning these tags to a single scan class, multiple scan classes are used. The user defines ten scan classes and assigns 1,000 tags to each one. A 10-minute update rate is chosen with each scan class offset by one minute. The result is each scan executes one minute after the next distributing the 10,000 tag updates into 10 requests of 1,000 tags each. When compared to having all tags belong to one scan class, this configuration is a more efficient use of server resources.

Scan Rates

The scan class update frequency can be chosen to minimize the performance impact on the source Historian Server. Recent archive data is cached in Historian Server memory for each tag. Historian Server resource usage is minimized when interface updates are obtained by reading cached data. If the data does not reside in memory it is read from disk. Disk reads require more hardware resources than a memory read. When possible a scan frequency should be chosen that minimizes disk access while maximizing the time between scans.

A Historian 3.4 server allocates space for 4 data events per point in the archive read memory cache. This cache size is a configurable parameter. When the interface performs history recovery nearly all data is obtained from disk reads. When the interface transitions to scanning for updates, disk reads should decrease as recent data is obtained from the archive memory cache. The percentage of data obtained via disk reads versus memory can be managed through the scan frequency. Increasing the scan rate should increase the percentage read from memory. Vice versa, decreasing the scan rate increases the percentages of data obtained from disk reads. With some trial and error users can determine a scan rate that maximizes the time between scans while minimizing archive disk reads. For Historian 3 source servers, use the `Historian Performance Monitor` interface to access archive cache versus disk read data counters. Historian 2 source servers will want to monitor disk versus cache read ratios.

Data Timestamps

The source Historian Server provides a timestamp for each data event. These timestamps may be adjusted to account for time zone differences and clock drift between Historian Servers. It is required that each Historian Server have the correct system time for the configured time zone.

Timestamps are automatically adjusted for time zone differences when both source and receiving servers are PI3. PI3 uses Universal Coordinated Time (UTC) to timestamp data. UTC is based on seconds since Jan.1, 1970 in Greenwich Mean Time. Each UTC time stamp contains a time offset from Greenwich Mean Time that represents the local time zone setting.

If source and/or receiving Historian Server is PI2, users have the option of adjusting for time zone differences. PI2 uses local time. Local time is seconds since Jan. 1, 1970 in the local time zone. PI2 data timestamps do not include a time zone offset. For example, viewing data from a PI2 server in a time zone two hours behind your local time will appear to be two hours old. PI3 data includes time zone information that allows for automatic adjustment to local time.

The interface can also adjust for clock drift between Historian Servers. Clock drift is the time offset between Historian Servers after accounting for time zone differences. An offset of 30 minutes or less is considered clock drift. When configured to do so, the time offset is added to the timestamp provided by the source Historian Server adjusting it to receiving Historian Server time.

Timestamp adjustment is configured on a tag-by-tag basis through the `Location2` tag attribute. Note that all computers (interface node, source and receiving Historian Server) must have the correct system time for the configured time zone.

Data Type Conversions

The following table displays supported data type conversions between source and receiving Historian tags.

	Receiving Tag Point Type				
Source Tag Point Type		**Float 16/32/64**	**Int16/32**	**Digital**	**String**
	Float 16/32/64	▨	Yes	Yes	No

	Int16/32	Yes	▨	Yes	No
	Digital	No	Yes	▨	No
	String	No	No	Yes	▨

Interface Status Events

The FactoryTalk Historian to Historian interface may update a tag to indicate a specific status event. These status events represent data that is generated by the interface and therefore will not exist for the source tag.

There are three specific status events generated by the interface:

- **IO Timeout** status events are triggered when the interface loses a Historian Server connection. This information informs users that current data is not being collected. An event is written to indicate the time of disconnection and another event is written to indicate the time of reconnection.
- **Access Denied** status events are written whenever the interface is denied access to a tag's data or attribute information.
- **Intf Shut** status events are enabled through **`/stopstat`** startup parameter. When enabled an event is written when the interface is stopped and again on startup. These events tell users no data is being collected because the interface is not running. See a more detailed description in the "Command-line Parameters" section.

To prevent a data mismatch between Historian Servers these status events should be disabled. If enabled these events create a data gap that will not be filled through history recovery.

Each tag receives an 'IO Timeout' event at the time of Historian Server disconnection and a second event at reconnection. Likewise, when the interface is stopped and started a 'Shutdown' event is written to each tag. History recovery begins either at the start of the recovery period or a tag's current value, whichever is more recent. The status events are updated prior to performing history recovery, making them the starting point of the recovery period for each tag. This results in a gap in data for a time period that might otherwise be recovered.

Interface status events are configured on a tag-by-tag basis through the `Location3` tag attribute. Interface shutdown events are enabled through the interface startup script. By default shutdown events are not written by the interface.

Adding, Removing and Editing Tags

Tag definitions can be modified while the interface is running. The interface periodically checks with the receiving Historian Server for tag configuration updates. These updates include adding, editing, and removing tags. During normal operation the interface checks for updates every two minutes. It will only process up to 25 tag updates on any given update. This is to prevent the processing of tag updates from delaying data collection. If there are more than 25 tag updates, the interface will check again every 30 seconds until all updates have been processed. Processing tag updates is a low priority for the interface. If a large number of tag edits are made, a user may choose to restart the interface. Restarting the interface will force it to process all edits by rebuilding its tag list.

Error Handling

When the interface encounters an error, it will print a message to the log file. If the error is tag-specific, the error message will include the tag name along with the specific error code. The tag is then marked by the interface as being in error and removed from the update list. The interface will attempt to clear tags in error every 10 minutes. It does this by trying to read a value from the source tag and then update the interface tag with that value. If this is successful, the tag is added back to its assigned scan class for data collection. Otherwise, it remains in error until the next attempt to clear it.

If an error is not tag-specific, the interface verifies server connectivity. Typically system errors are a result of network upsets. When a system error is encountered, the interface verifies Historian Server communication and attempts to continue collecting data.

Source Historian Server-Level Failover

Source Historian Server-level failover maximizes interface data availability on the receiving Historian Server. It requires that two Historian source servers are available that have identical tag definitions and data streams for interface points. This requirement ensures the interface will obtain the same data regardless of which source server is active. Source Historian Server-level failover is not supported for Historian 2.

Failover is enabled by specifying a primary and secondary source server in the interface startup file. On startup the interface attempts to establish a connection to each source server then load and initialize its tag list. Data collection begins from the primary source server making it the active node while the secondary source server assumes the standby role. However, if one of the source servers is unavailable on startup, the other server is designated the active source regardless of primary or secondary configuration.

Note: Source tag mapping by point ID is not compatible with server-level failover. This is because there is no guarantee of a point ID match between two source Historian Servers unless they are part of a Historian Collective.

Fault Conditions

There are two conditions that will cause the interface to initiate failover: a communication failure or detection of stale data.

Communication Failure

The first failover condition is when the interface loses communication to the active Historian source server. This may be the result of a temporary network upset, shutdown of the active source server, hardware failure, etc. The interface has no way of identifying what causes the disconnection. A disconnection means the interface did not receive a response from the active source within the timeout period. The interface may initiate a short wait then attempt to reconnect to the active source before attempting to failover. The timeout period, pause between reconnection, and number of reconnection attempts are all user configurable parameters.

Note: the default network timeout period is 60 seconds. This is not an interface configuration parameter. This is configured through the PI API configuration file, pilogin.ini. Please refer to the PI API Users Manual for information on adjusting the default timeout period.

Detection of Stale Data

The second failover condition is stale source data. The Historian Interface Status Utility is required on each source server to track data quality. This utility is a separate program that monitors updates to a specified watchdog tag. The utility updates a status tag to indicate whether or not the watchdog tag is being updated. The interface monitors this status tag as an indicator that source data is current. When configuring the Interface Status Utility users will want to select a watchdog tag that receives reliable high frequency updates.

Whenever possible the interface is connected to a source Historian Server that is actively receiving data. If this is not possible, the interface enters a loop where it waits for the first available source Historian Server receiving data.

Note: The Historian Interface Status Utility is not included as part of the interface installation. It is available for downloaded on our website; http://support.rockwellautomation.com

UniInt Failover

This interface supports UniInt Phase 2 failover with warm failover configuration. Refer to section UniInt Failover Configuration of this document for configuring the interface for failover.

Interface Status Tags

The interface has the optional functionality of outputting status data to a digital tag. There are two optional status tags that are mutually exclusive. These are digital tags that are created on the receiving Historian Server. These tags are useful for tracking performance and assisting with troubleshooting.

Internal State Status

The interface updates the internal state tag to indicate when it executes specific functions. Specifically, the status will indicate whether the interface is in startup, performing history recovery, scanning for data, experiencing network communication errors, denied data or point access (security permissions) or performing shutdown operations. This functionality is enabled by specifying an internal state status tag in the interface startup script.

Failover Status

When configured for server-level failover users can specify a status tag. The interface will update this tag to indicate whether the primary or secondary source server is active. It will also update this tag when it is in the process of failing over (no active source) or experiencing network communication errors.

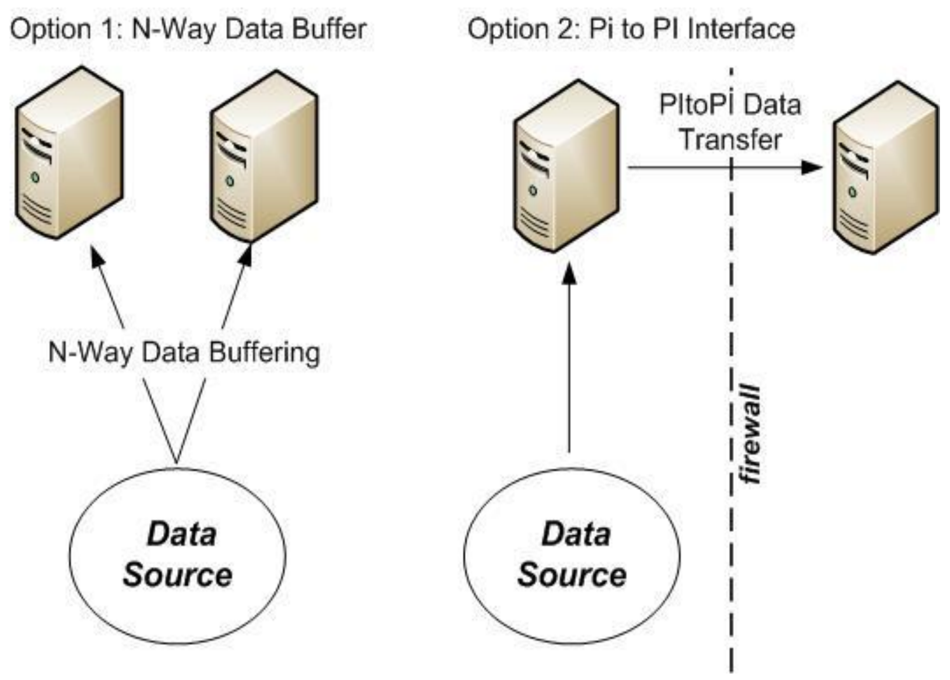
Deployment Scenarios

There are two typical FactoryTalk Historian to Historian interface applications. One is to isolate users from the source Historian Server. This may be for security requirements, load distribution, database centralization or to resolve remote access issues. In this scenario users will want to configure the interface for exception data collection to provide real-time updates to end users.

The other application is to maintain a copy of archive data on a secondary Historian Server. For example, in this scenario the receiving Historian Server maybe a backup of the source Historian Server. This is achieved by running the interface in archive data collection mode with tag attribute overrides in the startup configuration file. The tag attribute overrides enable users to configure interface data collection without configuring receiving tags specifically for PItoPI. This strategy allows receiving tags to be configured for their native interfaces and eliminates the need for mass tag editing prior to bringing the backup server online, reducing down time.

Historian to Historian within a Historian Collective

PItoPI can be used to distribute data within a collective. The primary method for data distribution within a collective is n-way data buffering from the data source. N-way data buffering provides the best performance and is the most efficient method for data distribution. However due to system architecture and security restrictions this may not always be possible.



Historian to Historian is used to transfer data within a collective when the data cannot be sent directly from the interface node. The following section describes typical configuration and deployment scenarios when running Historian to Historian within a collective.

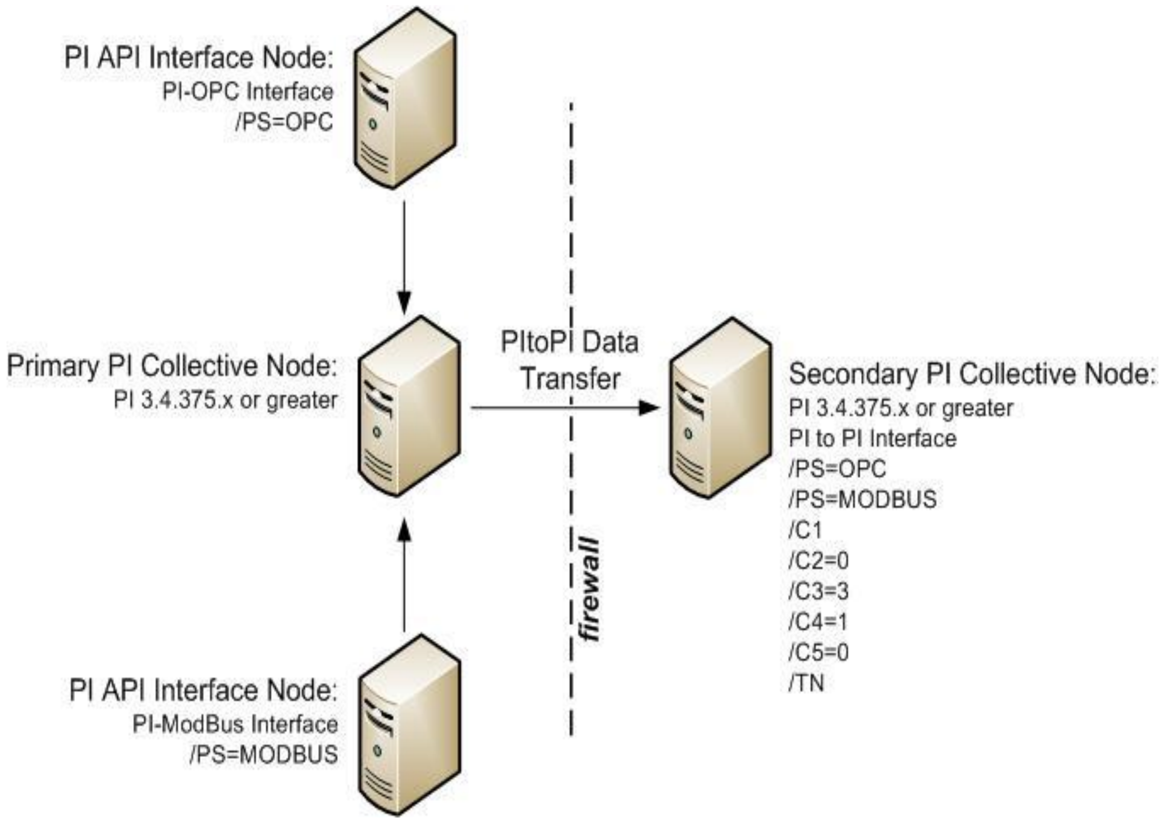
Tag Attribute Override Parameters

The interface supports tag attribute override parameters. These startup parameters must be used when running the interface within a collective. Tag attribute override parameters enable the interface to collect data for tags that are not actually configured for the Historian to Historian interface.

Each member of a collective is required to have identical Historian point database definitions. Using Historian to Historian within a collective will result in the need to have a tag configured for Historian to Historian on one collective member but these same tags must also be configured for the native data source interface on a different collective member.

For example, consider a secondary Historian Collective node that sits behind a firewall. The primary Historian Collective node receives data directly from a Historian OPC and a Historian ModBus interface node. The points on the secondary Historian Collective node are defined exactly as they are on the primary Historian Collective node. This means these points are either configured for the Historian OPC or the Historian ModBus interface

Example Architecture for Historian to Historian within a Historian Collective



Tag attribute override parameters allow users to pass global settings that enable the interface to identify and build its tag list without requiring the tags to be defined explicitly for Historian to Historian. A detailed description of the tag attribute override parameters can be found in section Startup Command File section of this manual.

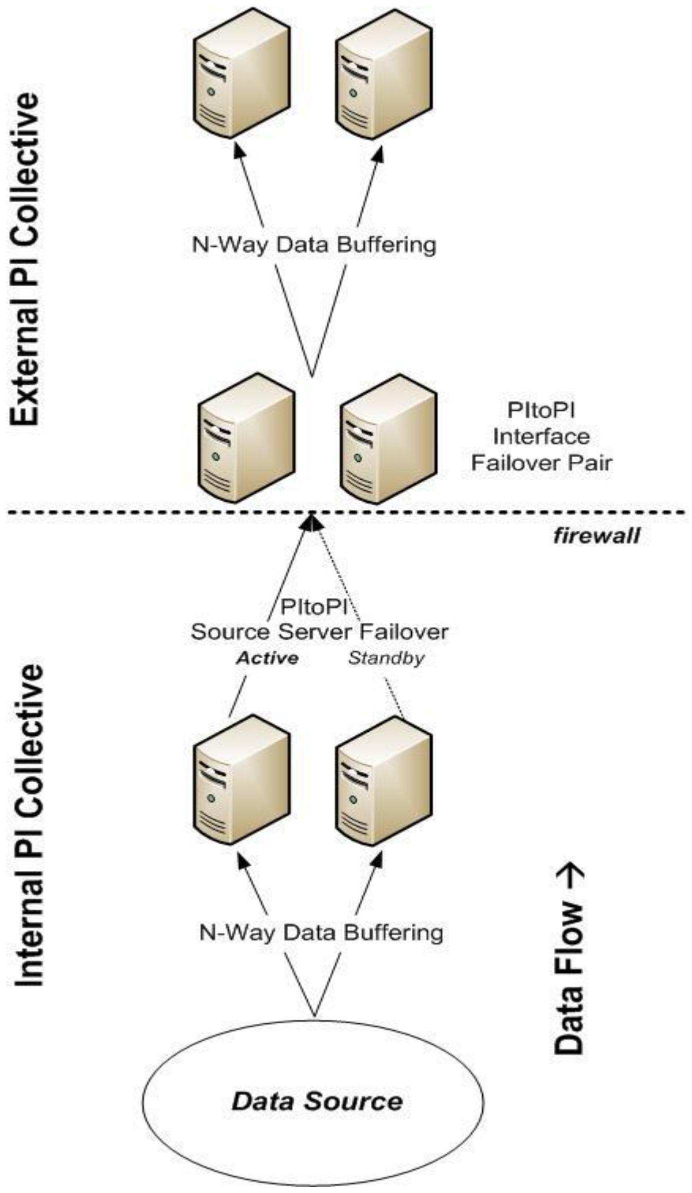
Firewall Considerations

When a firewall separates the source and receiving Historian Servers for the Historian to Historian interface users may want to run the interface outside of the firewall network. This configuration requires that port 5450 be opened for incoming connections to the Historian Server through the firewall. All outgoing connections from the Historian Server will require that port 1024+ and greater are opened. Additional configuration information for implementing a firewall can be found in our knowledge base (KB) article #2820OSI8 which is available on our TechSupport website (http://support.rockwellautomation.com).

For the best security posture, Historian to Historian should push data out from the most sensitive Historian Server. Enable buffering to eliminate the latency bottleneck.

Historian to Historian between Historian Collectives

The following diagram architecture depicts a typical deployment of Historian to Historian for aggregating data between two Historian Collectives.



Limitations

The following limitations apply to the Historian to Historian interface when aggregating data between two collectives.

Historian Collective Support

The Historian to Historian interface is not collective aware. While multiple Historian Servers will compose a collective, the interface will only know about the Historian Servers specified in its startup file. This means on startup both receiving and source Historian Server(s) specified in the interface startup file must be available in order for the interface to initialize its point list.

After the interface is up and running, and if n-way buffering is enabled, the interface will collect data even if the receiving Historian Server is not available. This means data flow is not dependent on the receiving Historian Server once the interface has completed is startup initializations.

Be aware that history recovery can be compromised when the interface is configured for n-way buffering to a collective. On startup the interface will check the snapshot value on the receiving Historian Server for each tag in its tag list. The snapshot value is used to determine the starting point for history recovery. This snapshot check only occurs on the receiving Historian Server specified in the interface startup file. Therefore if the last value for each tag assigned to the interface is not the same among Historian Servers in the receiving collective, it will result in either a data gap or data overlap. The best way to avoid this scenario is to initialize all Historian Servers in the receiving collective with the same set of data for the interface tag list prior to implementing the Historian to Historian interface.

Note: The Buffer Subsystem v3.4.375.38 only supports writing to the archive in NO REPLACE mode. This prevents the Historian to Historian interface from honoring Location5 archive write options and can lead to data loss. It is recommended users use a later version of Buffer Subsystem or Historian Bufserv to avoid this issue. Please check our support website for the latest versions; http://support.rockwellautomation.com

Data Latency and Source Historian Server Failover

The source Historian Server (or source Historian Server pair if failover is enabled) should receive data directly from the interface node populating data for the Historian to Historian source tag list. Not only will this ensure a seamless data transition on failover, it will also minimize data latency to the receiving Historian Server(s). In this context data latency is the time it takes between obtaining a value at the data source and having it arrive at the receiving Historian Server via the Historian to Historian interface.

UniInt Failover

This interface supports UniInt failover. Refer to the UniInt Failover Configuration section of this document for configuring the interface for failover.

Chapter 3. Installation Checklist

If you are familiar with running FactoryTalk Historian data collection interface programs, this checklist helps you get the interface running. If you are not familiar with Historian 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 node regardless of how many interfaces run on that node.

The Data Collection Steps below are required. Interface Diagnostics and Advanced Interface Features are optional.

Data Collection Steps

1. Confirm that you can use SMT to configure the Historian Server. You need not run SMT on the same computer on which you run this Interface.
2. If you are running the interface on an interface node, edit the Historian Server's Trust Table to allow the interface to write data.
3. Run the installation kit for the Historian Interface Configuration Utility (ICU) on the interface node if the ICU will be used to configure the interface. This kit runs the PI SDK installation kit, which installs both the PI API and the PI SDK.
4. Run the installation kit for this Interface. This kit also runs the PI SDK installation kit which installs both the PI API and the PI SDK if necessary.
5. 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 Historian Server to reject the data that this Interface writes.
6. Run the ICU and configure a new instance of this Interface. Essential startup parameters for this Interface are

   Point Source (`/PS=x`)
   Interface ID (`/ID=#`)
   Historian Server (`/Host=host:port`)
   Scan Class(`/F=##:##:##,offset`)
7. If you will use digital points, define the appropriate digital state sets on receiving node. These should correspond to digital state sets assigned to source tags.
8. Choose a point source. If Historian 2 home node, create the point source.

9. Build input tags and, if desired, output tags for this Interface. Important point attributes and their use are:

   `Location1` specifies the interface instance ID
   `Location2` specifies for timestamp adjustment.
   `Location3` specifies for configuring interface status events.
   `Location4` specifies the scan class.
   `Location5` sets the write mode of archived data on the receiving Historian Server.
   `InstrumentTag` specifies the name of the Source Tag from the Source Historian Server.
   `ExDesc` is an alternative way to specify the Source Tag. This can be specified with `STAG=<`*`tagname`*`>`.
   `UserInt1` defines Source Tag point ID instead of using source tag name.

10. Start the interface interactively and confirm its successful connection to the Historian Server without buffering.

11. Confirm that the interface collects data successfully.

12. Stop the interface and configure a buffering application (either Bufserv or PIBufss). When configuring buffering use the ICU menu item Tools → Buffering… → Buffering Settings to make a change to the default value (32678) for the Primary and Secondary Memory Buffer Size (Bytes) to 2000000. This will optimize the throughput for buffering and is recommended by OSIsoft.

13. Start the buffering application and the interface. Confirm that the interface works together with the buffering application by either physically removing the connection between the interface node and the Historian Server Node or by stopping the Historian Server.

14. Configure the interface to run as a Service. Confirm that the interface runs properly as a Service.

15. Restart the interface node and confirm that the interface and the buffering application restart.

Interface Diagnostics

1. Configure Scan Class Performance points.
2. Install the Historian Performance Monitor Interface (Full Version only) on the interface node.
3. Configure Performance Counter points.
4. Configure UniInt Health Monitoring points
5. Configure the I/O Rate point.
6. Install and configure the Interface Status Utility on the Historian Server Node.
7. Configure the Interface Status point.

Advanced Interface Features

Configure UniInt Failover; see that section in this document for details related to configuring the interface for failover.

Chapter 4. Interface Installation

Rockwell Automation recommends that interfaces be installed on interface nodes instead of directly on the Historian Server node. An interface node is any node other than the Historian Server node where the FactoryTalk Historian application Programming Interface (PI API) has been installed (see the PI API manual). With this approach, the Historian Server need not compete with interfaces for the machine's resources. The primary function of the Historian Server is to archive data and to service clients that request data.

After the interface has been installed and tested, Buffering should be enabled on the interface node. Buffering refers to either PI API Buffer Server (Bufserv) or the PI Buffer Subsystem (PIBufss). For more information about Buffering see the Buffering section of this manual.

In most cases, interfaces on interface nodes should be installed as automatic services. Services keep running after the user 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 Historian Server node. In this case, the typical procedure is to install the Historian Server as an automatic service and install the interface as an automatic service that depends on the PI Update Manager and PI Network Manager services. This typical scenario assumes that Buffering is not enabled on the Historian Server node. Bufserv can be enabled on the Historian Server node so that interfaces on the Historian Server node do not need to be started and stopped in conjunction with Historian, but it is not standard practice to enable buffering on the Historian Server node. The Buffer Subsystem can also be installed on the Historian Server. See the *UniInt Interface User Manual* for special procedural information.

Naming Conventions and Requirements

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

When Configuring the interface Manually

It is customary for the user to rename the executable and the startup command file when multiple copies of the interface are run. For example, `PItoPI1.exe` and `PItoPI1.bat` would typically be used for interface number 1, `PItoPI2.exe` and `PItoPI2.bat` 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

PIHOME Directory Tree

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.

For 32-bit operating systems a typical `pipc.ini` file contains the following lines:

```
[PIPC]
PIHOME=C:\Program Files\Rockwell Software\FactoryTalk
Historian\PIPC
```

For 64-bit operating systems a typical `pipc.ini` file contains the following lines:

```
[PIPC]
PIHOME=C:\Program Files (X86)\PIPC
```

The above lines define the root of the `PIHOME` directory on the C: drive. The `PIHOME` directory does not need to be on the C: drive. Rockwell Automation recommends using the paths shown above as the root `PIHOME` directory name.

Interface Installation Directory

The interface install kit will automatically install the interface to:

`PIHOME\Interfaces\ PItoPI\`

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

Note: All FactoryTalk Historian to Historian interface files are installed into `PIHOME\Interfaces\PItoPI`. If multiple copies of the interface are already installed with names other than PItoPI, they will not be upgraded. Upgrade of multiple interfaces should be done by manually copying the executable files and startup files.

Interface Installation Procedure

The FactoryTalk Historian to Historian Interface setup program uses the services of the Microsoft Windows Installer. Windows Installer is a standard part of Windows 2000 and greater operating systems. To install, run the appropriate installation kit.

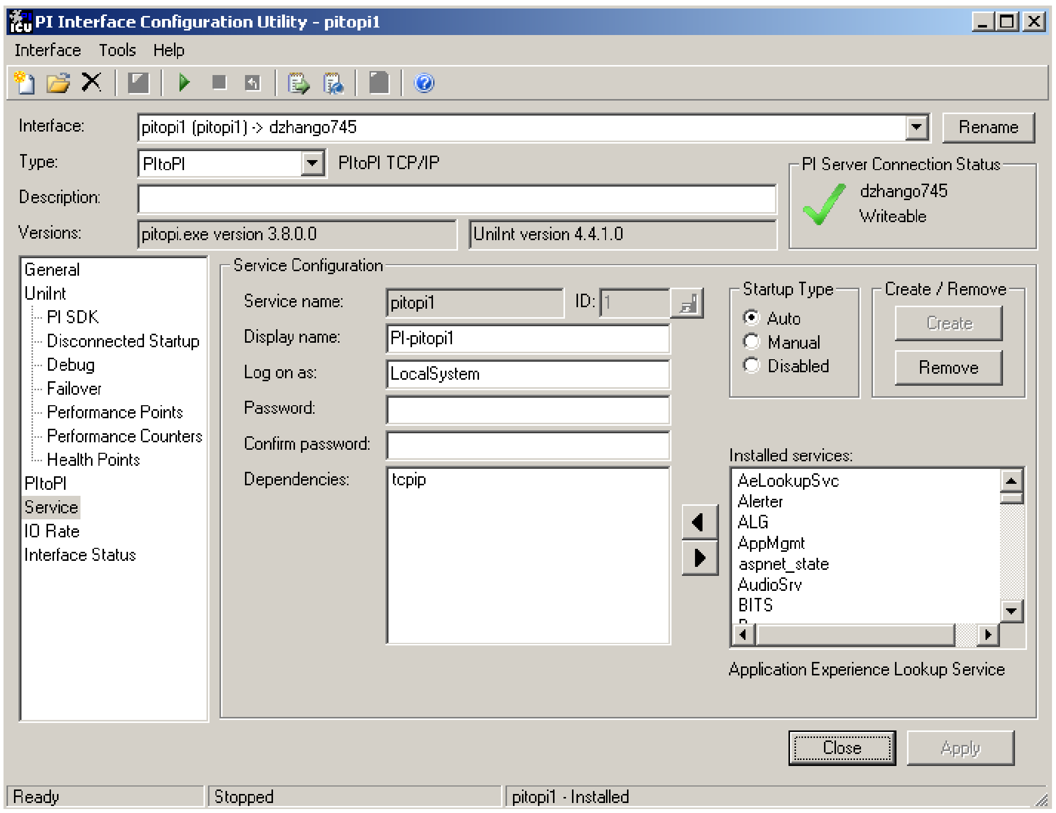
```
PItoPI_#.#.#.#_.exe
```

Installing Interface as a Windows Service

The FactoryTalk Historian to Historian Interface service can be created, preferably, with the Historian Interface Configuration Utility, or can be created manually.

Installing Interface Service with Historian Interface Configuration Utility

The Historian Interface Configuration Utility provides a user interface for creating, editing, and deleting the interface service:



Interface: pitopi1 (pitopi1) -> dzhango745
Type: PItoPI PItoPI TCP/IP
Description:
Versions: pitopi.exe version 3.8.0.0 UniInt version 4.4.1.0

PI Server Connection Status
dzhango745
Writeable

Service Configuration
Service name: pitopi1 ID: 1
Display name: PI-pitopi1
Log on as: LocalSystem
Password:
Confirm password:
Dependencies: tcpip

Startup Type
[✓ Auto] Manual Disabled

Installed services:
AeLookupSvc
Alerter
ALG
AppMgmt
aspnet_state
AudioSrv
BITS

Application Experience Lookup Service

Ready | Stopped | pitopi1 - Installed

Service Configuration

Service name

The *Service name* box shows the name of the current interface service. This service name is obtained from the interface executable.

ID

This is the service ID used to distinguish multiple instances of the same interface using the same executable.

Display name

The *Display Name* text box shows the current Display Name of the interface service. If there is currently no service for the selected interface, the default Display Name is the service name with a “Historian-” prefix. Users may specify a different Display Name. Rockwell Automation suggests that the prefix “Historian-” be appended to the beginning of the interface to indicate that the service is part of the Rockwell Automation suite of products.

Log on as

The *Log on as* text box shows the current “Log on as” Windows User Account of the interface service. If the service is configured to use the Local System account, the *Log on as* text box will show “LocalSystem.” Users may specify a different Windows User account for the service to use.

Password

If a Windows User account is entered in the *Log on as* text box, then a password must be provided in the *Password* text box, unless the account requires no password.

Confirm password

If a password is entered in the *Password* text box, then it must be confirmed in the *Confirm Password* text box.

Dependencies

The *Installed services* list is a list of the services currently installed on this machine. Services upon which this interface is dependent should be moved into the *Dependencies* list using the ◀ button. For example, if API Buffering is running, then “bufserv” should be selected from the list at the right and added to the list on the left. To remove a service from the list of dependencies, use the ▶ button, and the service name will be removed from the *Dependencies* list.

When the interface is started (as a service), the services listed in the dependency list will be verified as running (or an attempt will be made to start them). If the dependent service(s) cannot be started for any reason, then the interface service will not run.

Note: Please see the Historian Log and Windows Event Logger for messages that may indicate the cause for any service not running as expected.

◀ - *Add* Button

To add a dependency from the list of *Installed services*, select the dependency name, and click the *Add* button.

▶ - *Remove* Button

To remove a selected dependency, highlight the service name in the *Dependencies* list, and click the *Remove* button.

The full name of the service selected in the *Installed services* list is displayed below the *Installed services* list box.

Startup Type

The *Startup Type* indicates whether the interface service will start automatically or needs to be started manually on reboot.

- If the Auto option is selected, the service will be installed to start automatically when the machine reboots.
- If the Manual option is selected, the interface service will not start on reboot, but will require someone to manually start the service.
- If the Disabled option is selected, the service will not start at all.

Generally, interface services are set to start automatically.

Create

The *Create* button adds the displayed service with the specified *Dependencies* and with the specified *Startup Type*.

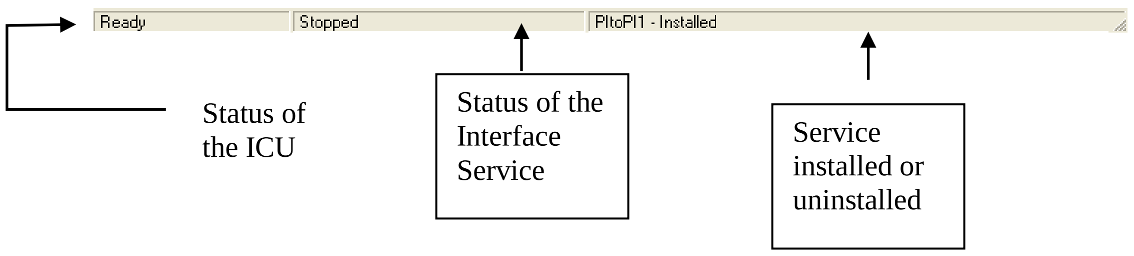
Remove

The *Remove* button removes the displayed service. If the service is not currently installed, or if the service is currently running, this button will be grayed out.

Start or Stop Service

The toolbar contains a *Start* button ▶ and a *Stop* button ■. If this interface service is not currently installed, these buttons will remain grayed out until the service is added. If this interface service is running, the *Stop* button is available. If this service is not running, the *Start* button is available.

The status of the interface service is indicated in the lower portion of the ICU dialog.


Ready
Stopped
PItoPI1 - Installed
Status of the ICU
Status of the Interface Service
Service installed or uninstalled

Installing Interface Service Manually

Help for installing the interface as a service is available at any time with the command:

```
PItoPI.exe -help
```

Open a Windows command prompt window and change to the directory where the `PItoPI1.exe` executable is located. Then, consult the following table to determine the appropriate service installation command.

Windows Service Installation Commands on an interface node or a Historian Server Node with Bufserv implemented	
Manual service	`PItoPI`.exe -install -depend "tcpip bufserv"
Automatic service	`PItoPI`.exe -install -auto -depend "tcpip bufserv"
*Automatic service with service ID	`PItoPI`.exe -serviceid X -install -auto -depend "tcpip bufserv"

Windows Service Installation Commands on an interface node or a Historian Server Node without Bufserv implemented	
Manual service	`PItoPI`.exe -install -depend tcpip
Automatic service	`PItoPI`.exe -install -auto -depend tcpip
*Automatic service with service ID	`PItoPI`.exe -serviceid X -install -auto -depend tcpip

*When specifying service ID, the user 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. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the Historian point as a point that belongs to a particular interface. For example, the string *Boiler1* may be used to identify points that belong to the *MyInt* Interface. To implement this, the PointSource attribute would be set to `Boiler1` for every Historian Point that is configured for the *MyInt* Interface. Then, if **`/ps=Boiler1`** is used on the startup command-line of the *MyInt* Interface, the interface will search the Historian Point Database upon startup for every Historian point that is configured with a PointSource of **`Boiler1.`** Before an interface loads a point, the interface usually performs further checks by examining additional Historian Point Attributes to determine whether a particular point is valid for the interface. For additional information, see the **`/ps`** parameter. If the PI API version being used is prior to 1.6.x or the Historian Server version is prior to 2.x, the Point**`Source`** is limited to a single character unless the SDK is being used.

Case-sensitivity for PointSource Attribute

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.

Chapter 6. Historian Point Configuration

The Historian point is the basic building block for controlling data flow to and from the Historian Server. A single point is configured for each measurement value that needs to be archived.

The following tag attributes are used for defining interface points on the receiving Historian Server.

Point Attributes

Use the point attributes below to define the Historian point configuration for the interface, including specifically what data to transfer.

Tag

The `Tag` attribute (or tagname) is the name for a point. There is a one-to-one correspondence between the name of a point and the point itself. Because of this relationship, FactoryTalk Historian documentation uses the terms “tag” and “point” interchangeably.

Follow these rules for naming Historian Points:

- The name must be unique on the Historian Server.
- The first character must be alphanumeric, the underscore (_), or the percent sign (%).
- Control characters such as linefeeds or tabs are illegal.
- The following characters also are illegal: * ’ ? ; { } [ ] | \ ` ‘ “

Length

Depending on the version of the PI API and the Historian Server, this Interface supports tags whose length is at most 255 or 1023 characters. The following table indicates the maximum length of this attribute for all the different combinations of PI API and Historian Server versions.

PI API	Historian Server	Maximum Length
1.6.0.2 or higher	2.x or higher	1023
1.6.0.2 or higher	Below 2.x	255
Below 1.6.0.2	2.x or higher	255
Below 1.6.0.2	Below 2.x	255

If the Historian Server version is earlier than 2.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum tag length of 1023, you need to enable the PI SDK. See Appendix_B for information.

The Historian 2 tag name attribute `LongName` is used if it exists.

The receiving and source tag names do not need to be identical. However, unless the receiving tag attributes `InstrumentTag` or `ExDesc` specify the source tag name or the **`/ptid`** interface parameter is used, the receiving tag name is assumed to be the same as the source tag name. The How PItoPI Finds Source Points section explains how the interface parameters affect which receiving tag attributes are used for mapping to the source point and the order that the attributes are searched for a mapping.

Note: If the source tag name length exceeds 80 characters, users must use the UserInt1 attribute for source point mapping. This is due to a limitation with the PI API programming library which supports tag names of 80 characters or less for point ID resolution.

PointSource

The PointSource is a unique, single or multi-character string that is used to identify the Historian point as a point that belongs to a particular interface. For additional information, see the **`/ps`** command-line parameter and the “PointSource” section.

PointType

The interface supports all Historian point types. Historian 2 point types are R (float16 and float32), D (digital) and I (int16). Historian 3 point types are float16, float32, float64, int16, int32, digital, timestamp, string, and blobs. For more information on the individual PointTypes, see Historian Server manuals.

Users should configure interface tags to be the same PointType as their source tag to maintain data consistency between Historian Servers. If interface tags have different PointType from their source tags, the interface will perform point type conversion. Refer to section Data Type Conversion of this document.

Location1

`Location1` indicates to which copy of the interface the point belongs. The value of this attribute must match the **`/id`** startup parameter.

In some cases users may run multiple copies of the interface that share the same point source. Each instance of the interface is differentiated by its interface ID. The `Location1` tag attribute defines which copy of the interface a tag is assigned to.

Location2

`Location2` specifies data timestamp adjustment. The source Historian Server supplies a timestamp for each data event. Users have the option of adjusting these timestamps to account for time differences between Historian Servers. It is required that each Historian Server have the correct system time for its configured time zone. See section Data Timestamps for a complete discussion.

Location2 Value	Adjust for Time Zone Differences	Do Not Adjust for Time Zone Differences*	Adjust for Clock Drift**	Truncate Sub-Second Timestamps***
0	Yes	-	--	--
1	--	Yes	--	--
2	Yes	--	Yes	--
3	--	Yes	Yes	--
4	Yes	--	--	Yes
5	--	Yes	--	Yes
6	Yes	--	Yes	Yes
7	--	Yes	Yes	Yes

* Only available if source and/or receiving Historian Server is a PI2 system.

** Receiving Historian Server is the time master. Timestamps are adjusted to receiving Historian Server time. An offset of 30 minutes or less is considered clock drift.

***Truncating sub-second timestamps can lead to data loss during history recovery and archive data scanning if multiple events are stored within the same second. Use with caution.

Location3

`Location3` is used to configure what interface status events are written to a tag.

‘IO Timeout’ status events will result in a data gap for periods of Historian Server disconnection. See section Interface Status Events for a complete discussion.

If a tag is configured for archive data collection, this attribute also specifies whether or not the snapshot value is included with each scan update.

Location3 Value	Write “I/O Timeout”	Write “Access Denied”	Include Snapshot	Point Level Debugging
0	Yes	Yes	--	--
1	--	Yes	--	--
2	Yes	--	--	--
3	--	--	--	--
4	Yes	Yes	Yes	--
5	--	Yes	Yes	--
6	Yes	--	Yes	--
7	--	--	Yes	--
8	Yes	Yes	--	Yes
9	--	Yes	--	Yes
10	Yes	--	--	Yes
11	--	--	--	Yes
12	Yes	Yes	Yes	Yes

Location3 Value	Write "I/O Timeout"	Write "Access Denied"	Include Snapshot	Point Level Debugging
13	--	Yes	Yes	Yes
14	Yes	--	Yes	Yes
15	--	--	Yes	Yes

Location4

Scan-based Inputs

For interfaces that support scan-based collection of data, `Location4` defines the scan class for the Historian point. The scan class determines the frequency at which input points are scanned for new values. For more information, see the description of the **`/f`** parameter in the Startup Command File section.

Location5

`Location5` attribute is used for setting the write mode for sending archive data to the receiving Historian Server. The following table lists the supported modes for PI3 and PI2 receiving Historian Servers.

Location5	Write Mode	Description	Supported Historian Server versions
0	ARCAPPEND Archive and Snapshot Data	Add archive event regardless of existing events. Default snapshot behavior is to append data if event at timestamp exists..	PI3 only*
1	ARCNOREPLACE Archive Events Only	Add archive event unless event(s) exist at same time. Default snapshot behavior is to append data if event at timestamp exists.	PI2 only
2	ARCREPLACE Archive Events Only	Add archive event, replace if event at same time. Default snapshot behavior is to append data if event at timestamp exists.	PI3 & PI2
3	ARCNOREPLACE Archive and Snapshot Events	Add archive or snapshot event unless event(s) exist at same time	PI2 only
4	ARCREPLACE Archive and Snapshot Events	Add archive or snapshot event, replace if event at same time	PI3 & PI2
*Note: PI2 supports Insert (ARCNOREPLACE) or Replace (ARCREPLACE) data write			

modes. It is not possible for PI2 to have two values with the same timestamp.

InstrumentTag

Note: If the source tag name length exceeds 80 characters, users must use the UserInt1 attribute for source point mapping. This is due to a limitation with the PI API programming library which supports tag names of 80 characters or less for point ID resolution.

Length

Depending on the version of the PI API and the Historian Server, this Interface supports an `InstrumentTag` attribute whose length is at most 32 or 1023 characters. The following table indicates the maximum length of this attribute for all the different combinations of PI API and Historian Server versions.

PI API	Historian Server	Maximum Length
1.6.0.2 or higher	2.x or higher	1023
1.6.0.2 or higher	Below 2.x	32
Below 1.6.0.2	2.x or higher	32
Below 1.6.0.2	Below 2.x	32

If the Historian Server version is earlier than 2.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum `InstrumentTag` length of 1023, you need to enable the PI SDK. See Appendix B for information.

Unless the **`/tn`**, **`/tnex`**, or **`/ptid`** interface parameter is used, the `InstrumentTag` can be used to specify the source tag. The `ExDesc` and `Tag` attributes can also be used for source tag name definition. The `UserInt1` attribute can be used to specify source tag by point ID instead of tag name. The How PItoPI Finds Source Points section explains how the interface parameters affect which receiving tag attributes are used for mapping to the source point and the order that the attributes are searched for a mapping.

The interface checks the `InstrumentTag` attribute first. If the `InstrumentTag` is empty, the extended descriptor is checked for `STAG=<`*`tagname`*`>` at the beginning of the `ExDesc` value. That is, the `ExDesc` attribute must begin with the ‘`STAG`’ keyword. If neither `InstrumentTag` nor `ExDesc` has a tag name, the `UserInt1` attribute is checked for the source tag point ID. Finally if none of these three attributes identify a source point, the receiving tag name is used for the source tag name.

ExDesc

Note: If the source tag name length exceeds 80 characters, users must use the UserInt1 attribute for source point mapping. This is due to a limitation with the PI API programming library which supports tag names of 80 characters or less for point ID resolution.

Length

Depending on the version of the PI API and the Historian Server, this Interface supports an `Extended Descriptor` attribute whose length is at most 80 or 1023 characters. The following table indicates the maximum length of this attribute for all the different combinations of PI API and Historian Server versions.

PI API	Historian Server	Maximum Length
1.6.0.2 or higher	2.x or higher	1023
1.6.0.2 or higher	Below 2.x	80
Below 1.6.0.2	2.x or higher	80
Below 1.6.0.2	Below 2.x	80

If the Historian Server version is earlier than 2.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum `ExDesc` length of 1023, you need to enable the PI SDK. See Appendix B for information.

Unless the **`/tn`** or **`/ptid`** interface parameter is used, the extended descriptor may be used as an alternative source tag specification by using the keyword `STAG=<`*`tag name`*`>`. The How PItoPI Finds Source Points section explains how the interface parameters affect which receiving tag attributes are used for mapping to the source point and the order that the attributes are searched for a mapping.

Note that the `STAG` keyword must be the first four characters of the `Exdesc` attribute value. No spaces can precede the `STAG` keyword. The `STAG` keyword is case insensitive. If additional information is included after the `STAG` specification, the tag name must be terminated with a comma or enclosed in double quotation marks.

If the source tag name contains a comma within it, it must be enclosed in double quotation marks. For example, `STAG="batchreactor1,temp"` is a legitimate way to define a source tag name.

Performance Points

For UniInt-based interfaces, the extended descriptor is checked for the string “PERFORMANCE_POINT”. If this character string is found, UniInt treats this point as a performance point. See the section called Scan Class Performance Points.

UserInt1

Note: If the source tag name length exceeds 80 characters, users must use the UserInt1 attribute for source point mapping. This is due to a limitation with the PI API programming library which supports tag names of 80 characters or less for point ID resolution.

Unless the **`/tn`** or **`/tnex`** interface parameter is used, the `UserInt1` attribute can be used to specify the source tag point ID. The How PItoPI Finds Source Points section explains how the interface parameters affect which receiving tag attributes are used for mapping to the source point and the order that the attributes are searched for a mapping.

If the source tag name is longer than 80 characters then you must use this attribute to specify the source tag point ID. This is an alternative to using the source tag name. The advantage of using a point ID is that a tag name can change but the point ID cannot. For example, if the source tag name is used and that name is changed, the interface point will no longer be able to find it and will therefore not be able to collect its data. Using the source tag point ID will prevent a tag rename from stopping data updates.

Scan

By default, the Scan attribute has a value of 1, which means that scanning is turned on for the point. Setting the Scan attribute to 0 turns scanning off. If the Scan attribute is 0 when the interface starts, a message is written to the `pipc.log` and the tag is not loaded by the interface. There is one exception to the previous statement.

If any Historian point is removed from the interface while the interface is running (including setting the Scan attribute to 0), SCAN OFF will be written to the Historian point regardless of the value of the Scan attribute. Two examples of actions that would remove a Historian point from an interface are to change the point source or set the Scan attribute to 0. If an interface specific attribute is changed that causes the tag to be rejected by the interface, SCAN OFF will be written to the Historian point.

Shutdown

It is recommended that shutdown events are disabled for all interface tags. There are two shutdown events; when the Historian Server is stopped and when the interface is stopped. The Shutdown tag attribute is used to enable Historian Server shutdown events. See section Interface Status Events for information regarding interface shutdown events.

Shutdown events are written to tags when the Historian Server is stopped to indicate data is not being collected. In some cases when the Historian Server is not running it creates a gap in data. However if interface history recovery is enabled a shutdown does not indicate a period of lost data.

If a tag is configured for shutdown events one is written when the server is stopped then again when it is started. Since the shutdown event on startup will be the current value for the interface tag it is used as the starting for history recovery. This results in a data gap for the Historian Server shutdown period.

Bufserv and PIBufss

It is undesirable to write shutdown events when buffering is being used. Bufserv and PIBufss are utility programs that provide the capability to store and forward events to a Historian Server, allowing continuous data collection when the Server is down for maintenance, upgrades, backups, and unexpected failures. That is, when Historian is shutdown, Bufserv or PIBufss will continue to collect data for the interface, making it undesirable to write `SHUTDOWN` events to the Historian Points for this Interface. Disabling Shutdown is recommended when sending data to a Highly Available Historian Server Collective. Refer to the Bufserv or PIBufss manuals for additional information.

Exception and Compression

Interface data may or may not be subjected to exception and compression depending on receiving Historian Server version and interface configuration. In either case it is recommended users configure the interface and tag attributes to prevent compression from causing data mismatches between Historian Servers.

Interface Configurations

Data collected by the interface has already passed exception on the source Historian Server. For this reason interface exception filtering should be disabled. This can be done by specifying the **`/sn`** switch in the startup configuration file. This tells the interface to bypass exception and send data directly to the snapshot table on the receiving Historian Server.

By default the interface will send history recovery and archive update data directly to the receiving Historian Server archive. The data write mode is configurable through the `Location5` tag attribute. If the **`/dc`** switch is specified in the interface startup file, data will pass through compression on the receiving Historian Server regardless of server version. If the receiving Historian Server is Historian 3.2 or Historian 2 data is always subjected to compression whether or not the **`/dc`** switch is used.

Note: if a tag configured for archive data updates has its `Location3` attribute set so it receives snapshot updates, data compression must be enabled to eliminate data mismatches between Historian Servers.

Recommended Tag Configurations

Each tag has attributes that are used to define exception and compression data filtering. These attributes specify a significant data change and the minimum and maximum time between events.

The following tag configurations are recommended to prevent data mismatches:

Tag Attribute	Value
ExcDev/ExcDevPercent	0
CompDev/CompDevPercent	Source Tag Value
ExcMin & CompMin	0
ExcMax & CompMax	32767

The `ExcDev/ExcDevPercent` and `CompDev/CompDevPercent` attributes define a significant data change. A new update must pass this value to pass data filtering.

The `ExcMax` and `CompMax` tag attributes set the maximum number of seconds between updates. When the value of this attribute is exceeded, the next update passes filtering regardless of whether or not it is a significant change in value. Tags should be configured with the maximum allowed values.

The `ExcMin` and `CompMin` tag attributes set the minimum number of seconds between updates. If a new value is received and the time difference between the new value and the current value does not exceed this value the update is filtered. Tags should be configured with a value of zero to prevent data from being filtered because of the time between updates.

By default, the largest `ExcMax` and `CompMax` values the interface supports is 32,767 seconds (~9hours). If the receiving Historian Server was a new installation of PI 3.3 or later, it is possible to have `ExcMax` and `CompMax` values of 4,294,967,295 seconds. In addition to the server version, the PI SDK must be enabled in the interface startup file by specifying **`/pisdk=1`**.

DataAccess, PtAccess

Historian 3 nodes use these attributes to control client read/write access to the data and point attributes. Access may be specified for owner, group and world. The user and groups used must be created before assignment. Example: `dataaccess=o:rw g:rw w:r`, `ptaccess=o:rw g:r w:r`.

Zero, Span

These attributes should match for the source and receiving points.

Chapter 7. Interface Status Tag Configuration

The following procedure describes how to create and configure interface status tags. These tags are created on the receiving Historian Server. See section Interface Status Tags for a complete description of this functionality.

1. Create a digital state set for the status tag. The following tables display the digital state set definitions.

   Internal Status Tag:

Digital State	String
0	STARTUP
1	HISTORYRECOVERY
2	SCANNING
3	SHUTDOWN
4	ACCESSDENIED
5	SRCDISCONNECT
6	RCVDISCONNECT

   Failover Status Tag:

Digital State	String
0	RECONNECTING
1	PRIMARY
2	FAILOVER
3	SECONDARY

2. Create a digital tag on the receiving Historian Server with the following attributes:

Attribute	Value
PointSource	L
PointType	Digital
Compressing	0
ExcDev	0
DigitalStateSet	As defined in the receiving PI3 server digital state database.

Chapter 8. Startup Command File

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

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.

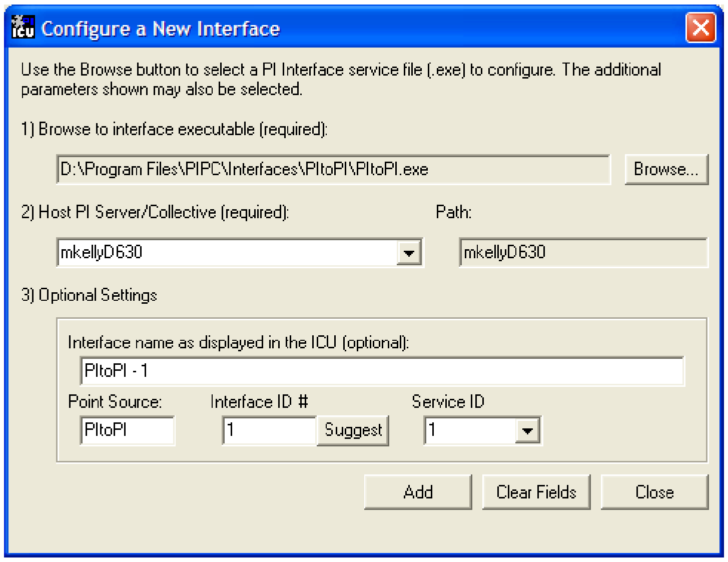
The Historian Interface Configuration Utility (ICU) provides a tool for configuring the interface startup command file.

Configuring the interface with ICU

Note: ICU requires PI 2.0 and later.

The Historian Interface Configuration Utility provides a graphical user interface for configuring Historian Interfaces. If the interface is configured by the ICU, the batch file of the interface (`PItoPI.bat`) will be maintained by the ICU and all configuration changes will be kept in that file and the module database. The procedure below describes the necessary steps for using ICU to configure the FactoryTalk Historian to Historian Interface.

From the ICU menu, select *Interface*, then *New Windows Interface Instance from EXE...*, and then *Browse* to the `PItoPI.exe` executable file. Then, enter values for *Point Source* and *Interface ID#*. A window such as the following results:



Use the Browse button to select a PI Interface service file (.exe) to configure. The additional parameters shown may also be selected.

1) Browse to interface executable (required):

D:\Program Files\PIPC\Interfaces\PItoPI\PItoPI.exe

2) Host PI Server/Collective (required): mkellyD630

Path: mkellyD630

3) Optional Settings

Interface name as displayed in the ICU (optional): PItoPI - 1

Point Source: PItoPI

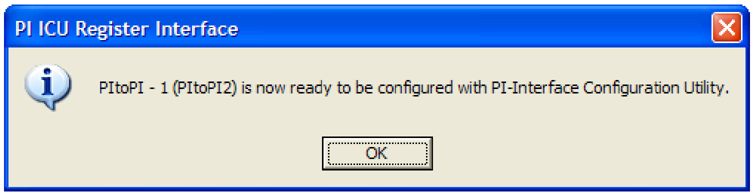
Interface ID #: 1

Service ID: 1

Interface name as displayed in the ICU (optional) will have Historian- pre-pended to this name and it will be the display name in the services menu.

Click *Add*.

The following display should appear:

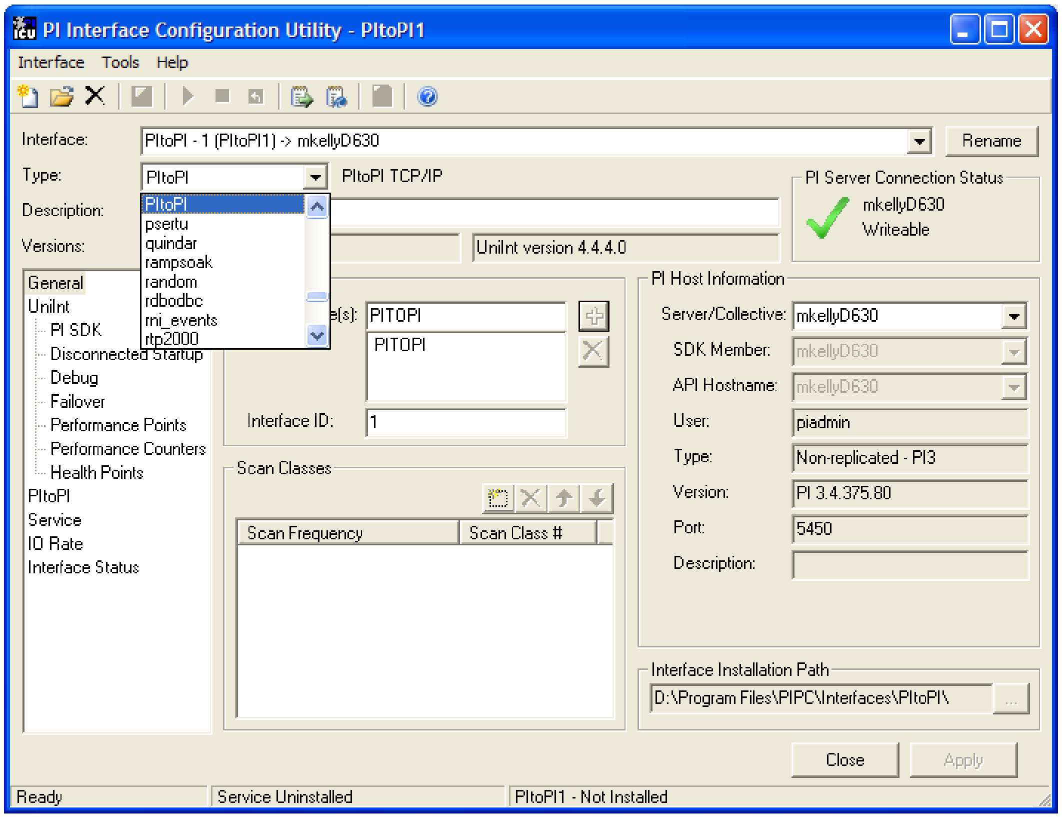

PI ICU Register Interface

PItoPI - 1 (PItoPI2) is now ready to be configured with PI-Interface Configuration Utility.

Note that in this example the Host FactoryTalk Historian System is mkellyD630. To configure the interface to communicate with a remote Historian Server, select '*Interface => Connections…*' item from ICU menu and select the default server. If the remote node is not present in the list of servers, it can be added.

Once the interface is added to ICU, near the top of the main ICU screen, the interface *Type* should be `PItoPI`. If not, use the drop-down box to change the interface *Type* to be `PItoPI`

Click on *Apply* to enable the ICU to manage this copy of the FactoryTalk Historian to Historian Interface.


PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI   PItoPI TCP/IP

Description:

Versions: UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

...e(s): PITOPI

PITOPI

Interface ID: 1

PI Host Information

Server/Collective: mkellyD630

SDK Member: mkellyD630

API Hostname: mkellyD630

User: piadmin

Type: Non-replicated - PI3

Version: PI 3.4.375.80

Port: 5450

Description:

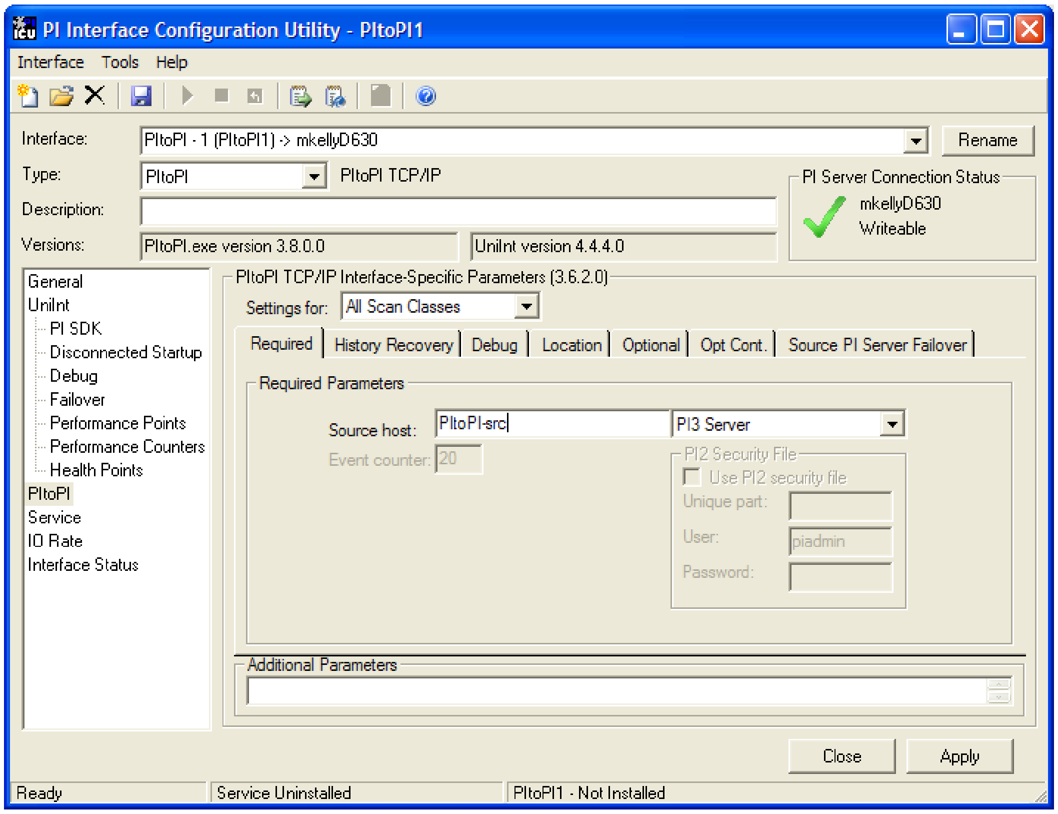
Interface Installation Path: D:\Program Files\PIPC\Interfaces\PItoPI\

Ready | Service Uninstalled | PItoPI1 - Not Installed

The next step is to make selections in the interface-specific page (that is, *PItoPI*) that allow the user to enter values for the startup parameters that are particular to the FactoryTalk Historian to Historian Interface.

The PItoPI ICU control has 7 tabs for each scan class, plus the All Scan Classes settings. A yellow text box indicates that an invalid value has been entered, or that a required value has not been entered.

The next figure shows the All Scan Classes settings.


PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI   PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0   UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

Required Parameters

Source host: PItoPI-src   PI3 Server

Event counter: 20

PI2 Security File

Use PI2 security file

Unique part:

User: piadmin

Password:

Additional Parameters

Ready | Service Uninstalled | PItoPI1 - Not Installed

Notice the box for the *Source host*. This parameter is required and must be replaced with your server name, where the Historian to Historian data will be retrieved from.

The *Settings for* drop down list will always have one entry titled “All Scan Classes”, and will have one entry for each scan class defined for this interface. The settings defined on the “All Scan Classes” tab are the settings that are used by the interface if one or more settings for any of the scan classes is not provided.

Since the FactoryTalk Historian to Historian Interface is a UniInt-based interface, in some cases the user will need to make appropriate selections in the UniInt pages. These pages allows the user to access UniInt features through the ICU and to make changes to the behavior of the interface.

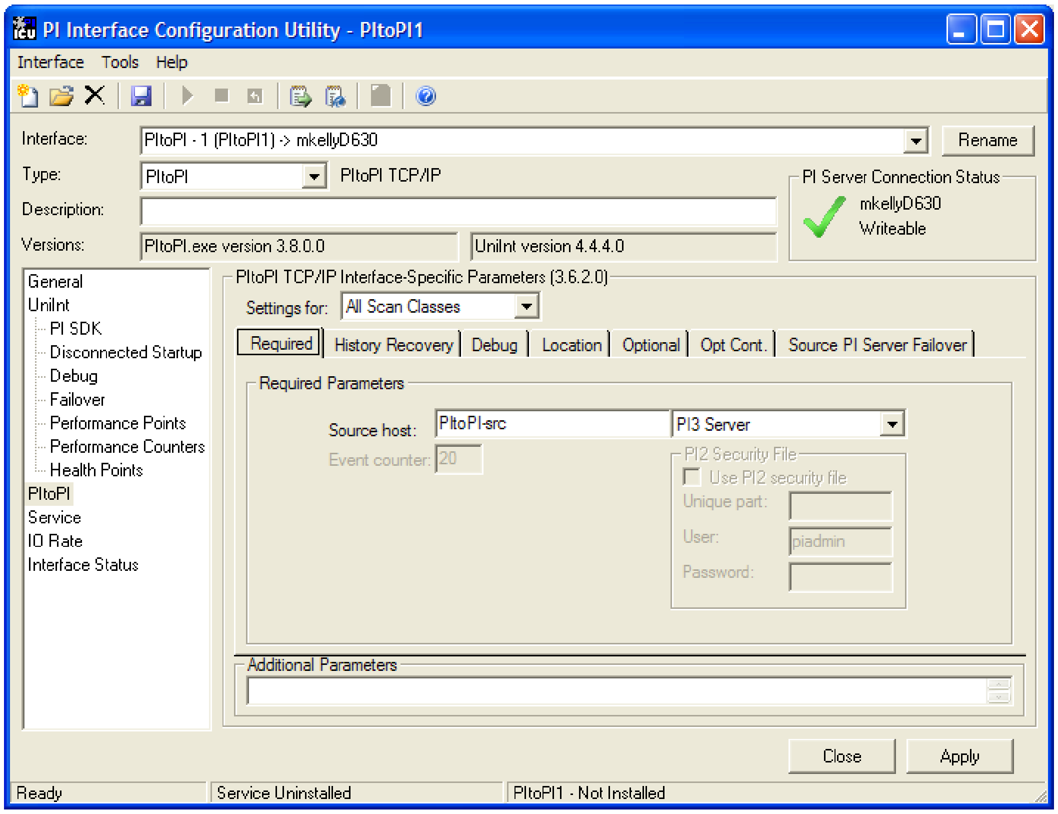
To set up the interface as a Windows Service, use the *Service* page. This page allows configuration of the interface to run as a service as well as to starting and stopping of the interface. The interface can also be run interactively from the ICU. To do that, open the *Interface* menu and then click *Start Interactive*.

For more detailed information on how to use the above-mentioned and other ICU pages and selections, please refer to the *Historian Interface Configuration Utility User Manual*. The next section describes the selections that are available from the *PItoPI* page. Once selections have been made on the ICU GUI, press the *Apply* button in order for ICU to make these changes to the interface’s startup file.

FactoryTalk Historian to Historian Interface page

Since the startup file of the FactoryTalk Historian to Historian Interface is maintained automatically by the ICU, use the *PItoPI* page to configure the startup parameters and do not make changes in the file manually. The following is the description of interface configuration parameters used in the ICU Control and corresponding manual parameters.

Required/General Tab


PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630
Type: PItoPI   PItoPI TCP/IP
Description:
Versions: PItoPI.exe version 3.8.0.0   UniInt version 4.4.4.0
PI Server Connection Status: mkellyD630 Writeable

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)
Settings for: All Scan Classes
Tabs: Required | History Recovery | Debug | Location | Optional | Opt Cont. | Source PI Server Failover

Required Parameters
Source host: PItoPI-src   PI3 Server
Event counter: 20
PI2 Security File: Use PI2 security file; Unique part: ; User: piadmin; Password:

Additional Parameters:

Status bar: Ready | Service Uninstalled | PItoPI1 - Not Installed

When **All Scan Classes** is selected in the *Settings for* list, the first tab will be titled Required. This is because the information provided in this first tab is required only for the All Scan Classes, and not for each scan class.

Required Parameters - Source host

The *Source host* is the name of the source Historian Server from which this FactoryTalk Historian to Historian interface is to get its data. (**`/SRC_HOST=hostname`**).

Required Parameters - PIx Server

The type of server needs to be selected in the combo box next to the *Source host* text box. The options are PI3 Server or PI2 Server.

Required Parameters - Event counter

The *Event counter* can only be configured for a particular scan class, so the *Event counter* box will remain disabled unless a scan class is selected in the *Settings for* combo box. (**`/EC=x`**).

PI2 Security File – Use PI2 security file

The *PI2 Security File* section is used only if the source host is a PI2 server. If a security file is to be used, select the *Use PI2 security file* check box.

PI2 Security File - Unique Part

The *Unique part* is the name suffix of tag security file on a Historian 2 system. The full name on the Historian 2 system is `PItoPI<name>`.SEC, where <*name*> is the portion specified in the *Unique part* box. (**`/SF=uniquename`**).

PI2 Security File - User

The *User* is the login user name of a PI user on Historian 2 node that the interface is to use. This is used for Historian 2 source systems. (**`/LN=username`**).

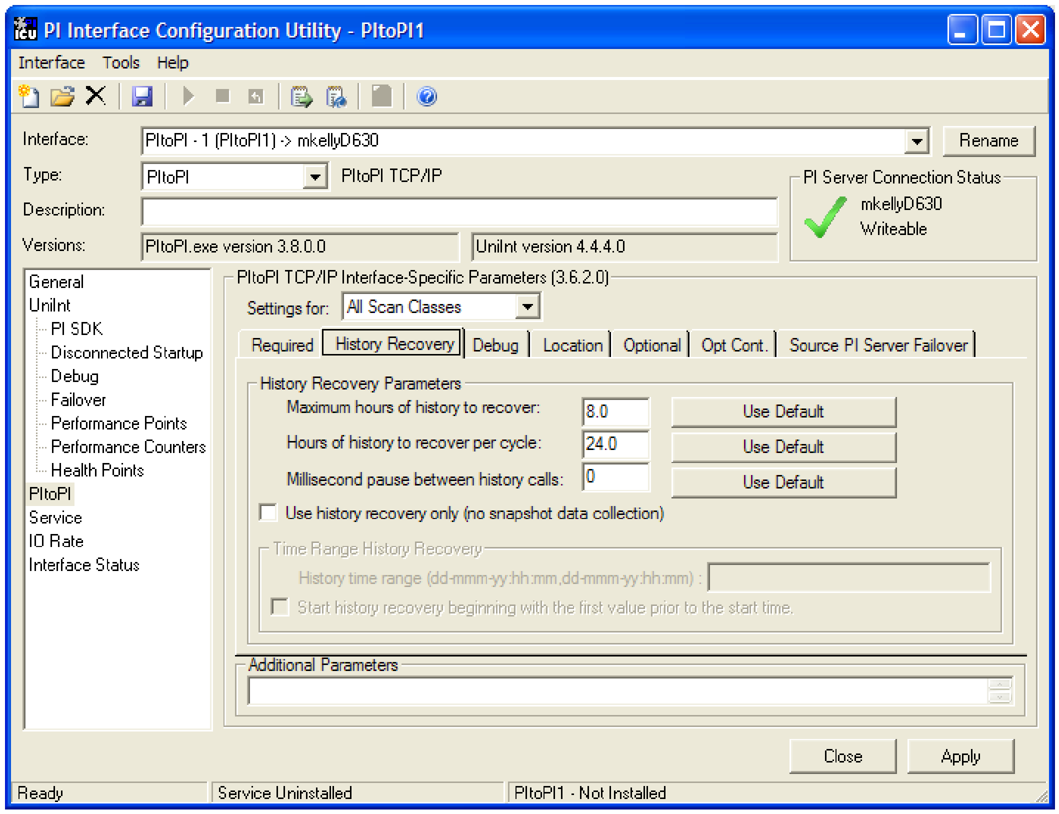
PI2 security File - Password

The *Password* is the login password of the PI user specified in the *User* box on Historian 2 node. This is used for Historian 2 source systems. (**`/PW=password`**).

Additional Parameters

This section is provided for any additional parameters that the current ICU Control does not support.

History Recovery Tab



PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0 UniInt version 4.4.4.0

PI Server Connection Status
mkellyD630
Writeable

General
UniInt
PI SDK
Disconnected Startup
Debug
Failover
Performance Points
Performance Counters
Health Points
PItoPI
Service
IO Rate
Interface Status

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

History Recovery Parameters

Maximum hours of history to recover: 8.0

Hours of history to recover per cycle: 24.0

Millisecond pause between history calls: 0

Use history recovery only (no snapshot data collection)

Time Range History Recovery

History time range (dd-mmm-yy:hh:mm,dd-mmm-yy:hh:mm) :

Start history recovery beginning with the first value prior to the start time.

Additional Parameters

Ready | Service Uninstalled | PItoPI1 - Not Installed

Maximum hours of history to recover

Number of hours to recover history for all points. Setting the value to 0 disables history recovery for all points. See section History Recovery for more information about history recovery. (**`/RH=hours`**)

The *Use Default* button is used to reset the value to the default setting of 8 maximum hours of history to recover.

Hours of history to recover per cycle

This is the number of hours of history to recover in each cycle through the point list. If the number of hours specified in the *Hours of history to recover per cycle* box is greater than or equal to the hours of history recovery requested in the *Maximum hours of history to recover* box, history will be recovered in one archive call from *- *Hours of history to recover per cycle* hours to *. If *Maximum hours of history to recover* is greater than *Hours of history to recover per cycle*, the archive calls to retrieve history will be divided into N calls, where N = *Maximum hours of history to recover* / *Hours of history to recover per cycle* + 1. The calls, which start from (*-*Maximum hours of history to recover*), will each span *Hours of history to recover per cycle* hours. Each history increment is collected for all tags in the given scan class before the next time increment is begun. If this field is set to zero, the default 24 hours will be used. (**`/RH_INC=hours`**)

The *Use Default* button is used to reset the value to the default setting of 24 hours for the number of *Hours of history to recover per cycle*.

Millisecond pause between history calls

The number of milliseconds to pause between history recovery calls. (**`/HRPAUSE=millisecond`**)

The *Use Default* button is used to reset the value to the default setting of 0 milliseconds to pause between history recovery calls.

Use history recovery only (no snapshot data collection)

If this check box is selected, tags do not sign up for exceptions. Each scheduled scan time (each scan class), history recovery is done from the last snapshot value to the current time. This box must be checked if you want to enter a *History time range*. (**`/HRONLY`**)

History time range (dd-mmm-yy:hh:mm:ss,dd-mmm-yy:hh:mm:ss)

Alternately, specifies a range of history to recover before exiting. The times must be specified using Historian Time string formats with a colon separating the date and the time. For example:

```
10-dec-98:10:00:00,10-dec-98:12:00:00
```

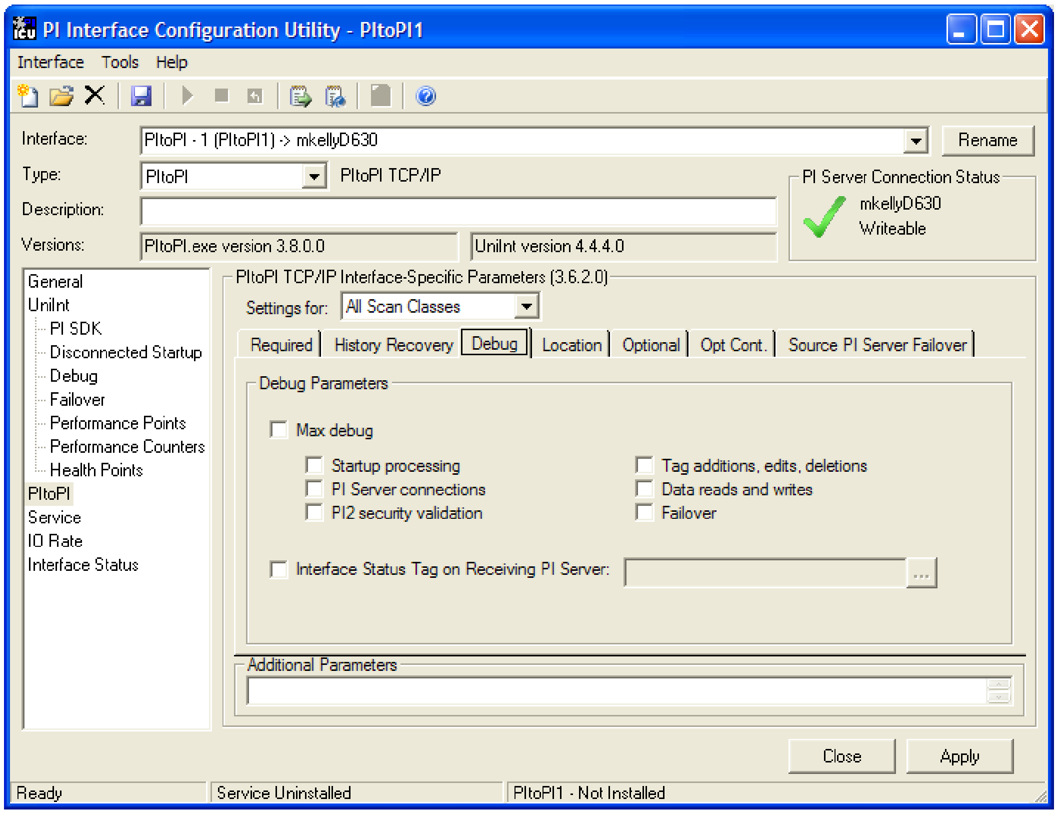
Note that these times are local to where the interface runs.

This will recover two hours of data from the source to receiving system; put it into the receiving FactoryTalk Historian System snapshot for all points and then exit. This switch will override the normal checking for the most recent snapshot time in the receiving database, thus out of order data may result. When time-range history recovery is enabled, the value specified by the **`/RH`** parameter is overridden. (**`/HRONLY=dd-mmm-yy:hh:mm:ss,dd-mmm-yy:hh:mm:ss`**)

Start history recovery beginning with the first value prior to the start time.

This will retrieve history for all the points starting from the value immediately prior to the start time. The default is to begin with the first value after the start time. This can only be checked if the *Use history recovery only* check box has been checked.

Debug Tab



PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI   PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0   UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

Debug Parameters

- [ ] Max debug
- [ ] Startup processing
- [ ] PI Server connections
- [ ] PI2 security validation
- [ ] Tag additions, edits, deletions
- [ ] Data reads and writes
- [ ] Failover
- [ ] Interface Status Tag on Receiving PI Server:

Additional Parameters

Ready | Service Uninstalled | PItoPI1 - Not Installed

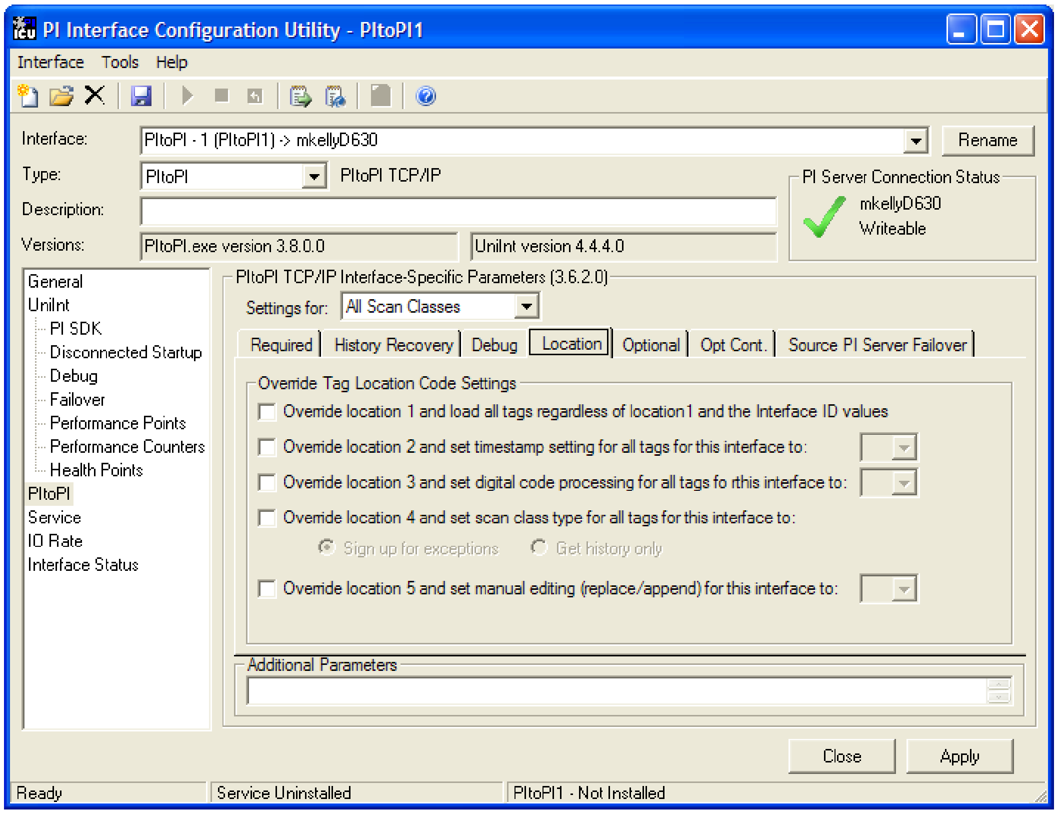
Debug Parameters

The Debug Levels parameter is used to set a debug level for debug messaging per scan class. Check all types of debug messages that you would like to see logged. Any combination of debug levels can be applied. (**`/DB=#,#,#,#...`**)

Interface Status Tag on Receiving Historian Server:

This is the name of an interface status tag configured on receiving server. Click the *Browse* button to browse the point database for this interface status tag using the Tag Search utility. (**`/IST=tagname`**)

Location Tab


PI Interface Configuration Utility - PItoPI1
Interface Tools Help
Interface: PItoPI - 1 (PItoPI1) -> mkellyD630
Rename
Type: PItoPI
PItoPI TCP/IP
PI Server Connection Status
mkellyD630
Writeable
Description:
Versions: PItoPI.exe version 3.8.0.0
UniInt version 4.4.4.0
General
UniInt
PI SDK
Disconnected Startup
Debug
Failover
Performance Points
Performance Counters
Health Points
PItoPI
Service
IO Rate
Interface Status
PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)
Settings for: All Scan Classes
Required
History Recovery
Debug
Location
Optional
Opt Cont.
Source PI Server Failover
Override Tag Location Code Settings
Override location 1 and load all tags regardless of location1 and the Interface ID values
Override location 2 and set timestamp setting for all tags for this interface to:
Override location 3 and set digital code processing for all tags fo rthis interface to:
Override location 4 and set scan class type for all tags for this interface to:
Sign up for exceptions
Get history only
Override location 5 and set manual editing (replace/append) for this interface to:
Additional Parameters
Close
Apply
Ready
Service Uninstalled
PItoPI1 - Not Installed

Use the *Override Tag Location Code Settings* check boxes to configure the interface to ignore individual tag location codes and to apply specific settings for each of the location codes.

Override Location 1

Ignore `Location1` for each tag and load all tags configured for the specified point source regardless of `Location1` and interface ID values. (**`/C1`**)

Override Location 2

Ignore `Location2` for each tag and set `Location2` value to be this number for all interface tags. (**`/C2=x`**)

Override Location 3

Ignore `Location3` for each tag and set `Location3` value to be this number for all interface tags. (**`/C3=x`**)

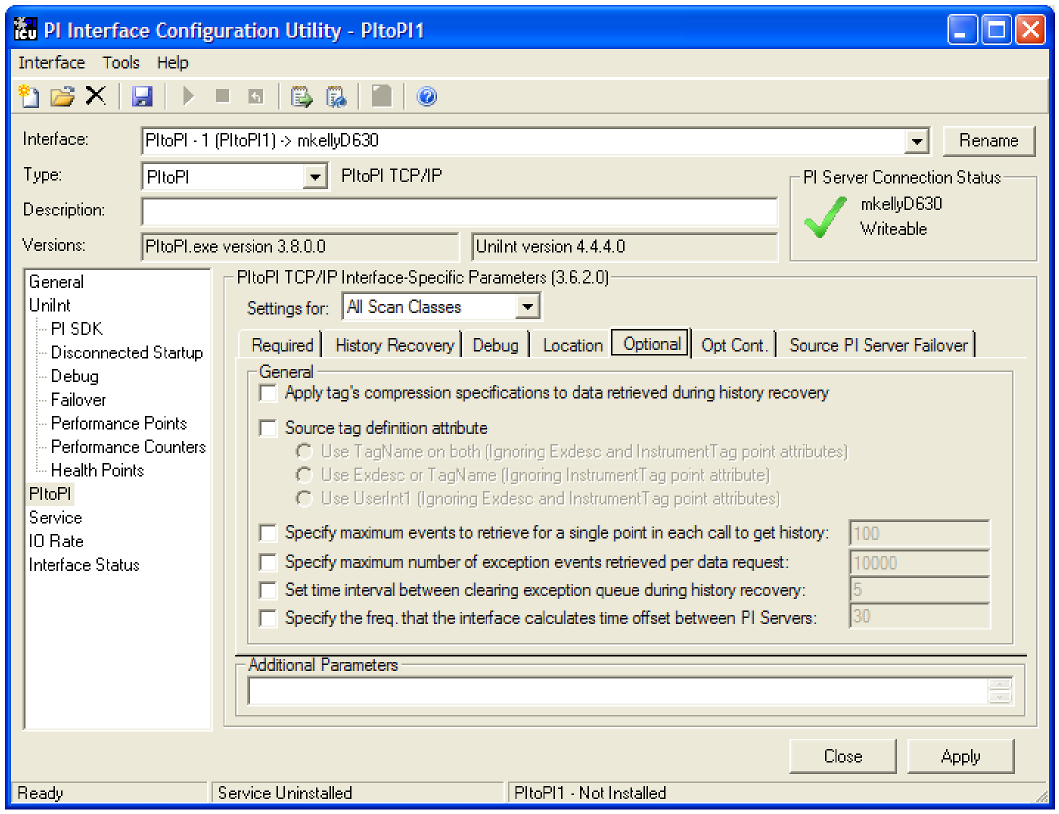
Override Location 4

Ignore `Location4` for each tag. The value used here should be 1, to have all points for the interface sign up for exceptions, or 2, to have all points retrieve history only. (**`/C4=x`**)

Override Location 5

Ignore `Location5` for each tag and set all points for the interface to the same value of this parameter (i.e., 0, 1, 2, or 3). (**`/C5=x`**)

Optional Tab


PI Interface Configuration Utility - PItoPI1

Interface Tools Help

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630 | Rename

Type: PItoPI | PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0 | UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

General
UniInt
- PI SDK
- Disconnected Startup
- Debug
- Failover
- Performance Points
- Performance Counters
- Health Points

PItoPI
Service
IO Rate
Interface Status

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

Required | History Recovery | Debug | Location | Optional | Opt Cont. | Source PI Server Failover

General
- [ ] Apply tag's compression specifications to data retrieved during history recovery
- [ ] Source tag definition attribute
  - ○ Use TagName on both (Ignoring Exdesc and InstrumentTag point attributes)
  - ○ Use Exdesc or TagName (Ignoring InstrumentTag point attribute)
  - ○ Use UserInt1 (Ignoring Exdesc and InstrumentTag point attributes)
- [ ] Specify maximum events to retrieve for a single point in each call to get history: 100
- [ ] Specify maximum number of exception events retrieved per data request: 10000
- [ ] Set time interval between clearing exception queue during history recovery: 5
- [ ] Specify the freq. that the interface calculates time offset between PI Servers: 30

Additional Parameters

Close | Apply

Ready | Service Uninstalled | PItoPI1 - Not Installed

Apply tag's compression specifications to data retrieved during history recovery.

Use compression specifications in tag configurations to send data retrieved during history recovery with compression. Usually data is retrieved from source server and sent to receiving server without compression during history recovery. (**`/DC`**)

Source tag definition attribute.

Use TagName on both (Ignoring ExDesc and InstrumentTag point attributes). Do not check the `InstrumentTag`, `ExDesc`, or `UserInt1` attributes for source tag definitions. Use `TagName` to identify the point on both source and receiving Historian Servers. (**`/TN`**)

Use ExDesc or TagName (Ignoring InstrumentTag point attribute). The source tag definition will be found in the `ExDesc` or `TagName` attribute. Ignore the `InstrumentTag` and `UserInt1` attributes. This parameter is useful for PI2 to PI3 migrations. (**`/TNEX`**)

Use UserInt1 (Ignoring ExDesc and InstrumentTag point attributes). The source tag definition will be found in the `UserInt1` attribute. Ignore the `ExDesc`, `InstrumentTag`, and `Tag` point attributes. (**`/PTID`**)

The How PItoPI Finds Source Points section explains how the interface parameters affect which receiving tag attributes are used for mapping to the source point and the order that the attributes are searched for a mapping.

Specify maximum events to retrieve for a single point in each call to get history.

This parameter is available for PI 2.0 and later receiving servers. It sets the maximum number of events to retrieve for a single point in each call to get history. With each call to retrieve history, one call is made to put it into the receiving server. At least one of these calls will be over the network, so using a small number could result in performance problems. (**`/MH=x, default: 1000`**)

Specify maximum number of exception events retrieved per data request.

This parameter sets the maximum number of exceptions events retrieved per data request. A large count reduces the number of calls required for acquiring exception updates. A small count reduces the time to complete each request (for troubleshooting network timeout issues). (**`/ME=#, default: 5000`**)

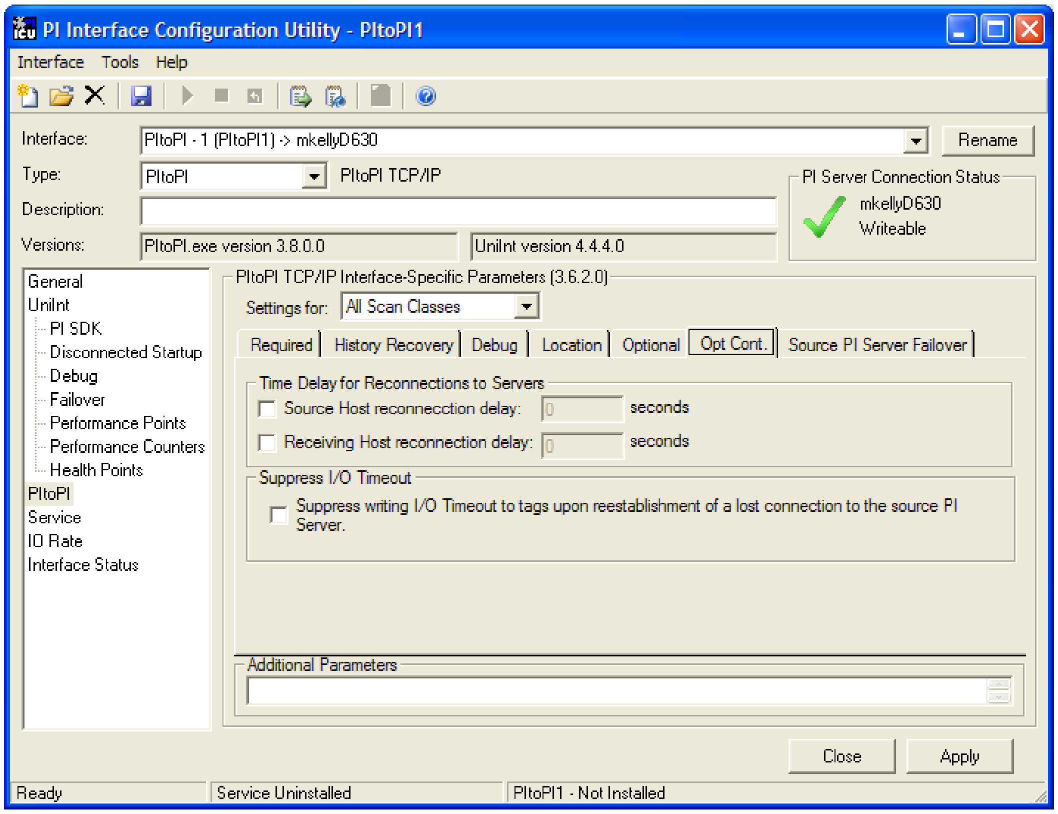
Set time interval between clearing exception queue during history recovery.

This parameter sets the time interval between clearing the exception queue on the source Historian Server for exception data scan classes. By default the interface will collect exceptions from the source Historian Server every 5 seconds during history recovery to prevent overflowing the queue. Users may want to adjust this time interval to tune history recovery performance. (**`/RH_QCKECK=#, default: 5`**)

Specify the frequency that the interface calculates time offset between Historian Servers.

This parameter sets the frequency in seconds at which the interface will calculate time offsets between Historian Servers. By default the interface will calculate time offsets every 30 seconds. (**`/OC=#, default: 30`**)

Opt Cont Tab



Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0 UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

Time Delay for Reconnections to Servers

- [ ] Source Host reconnecction delay: 0 seconds
- [ ] Receiving Host reconnection delay: 0 seconds

Suppress I/O Timeout

- [ ] Suppress writing I/O Timeout to tags upon reestablishment of a lost connection to the source PI Server.

Additional Parameters

Ready | Service Uninstalled | PItoPI1 - Not Installed

Source Host reconnection delay.

This parameter sets the time delay for attempting to reconnect to source Historian Server after a disruption. The number is entered in seconds and converted to milliseconds before being saved in the batch file. This number must be between 1 second and 8 hours. (**`/DELAYS=x, default: 0 seconds`**)

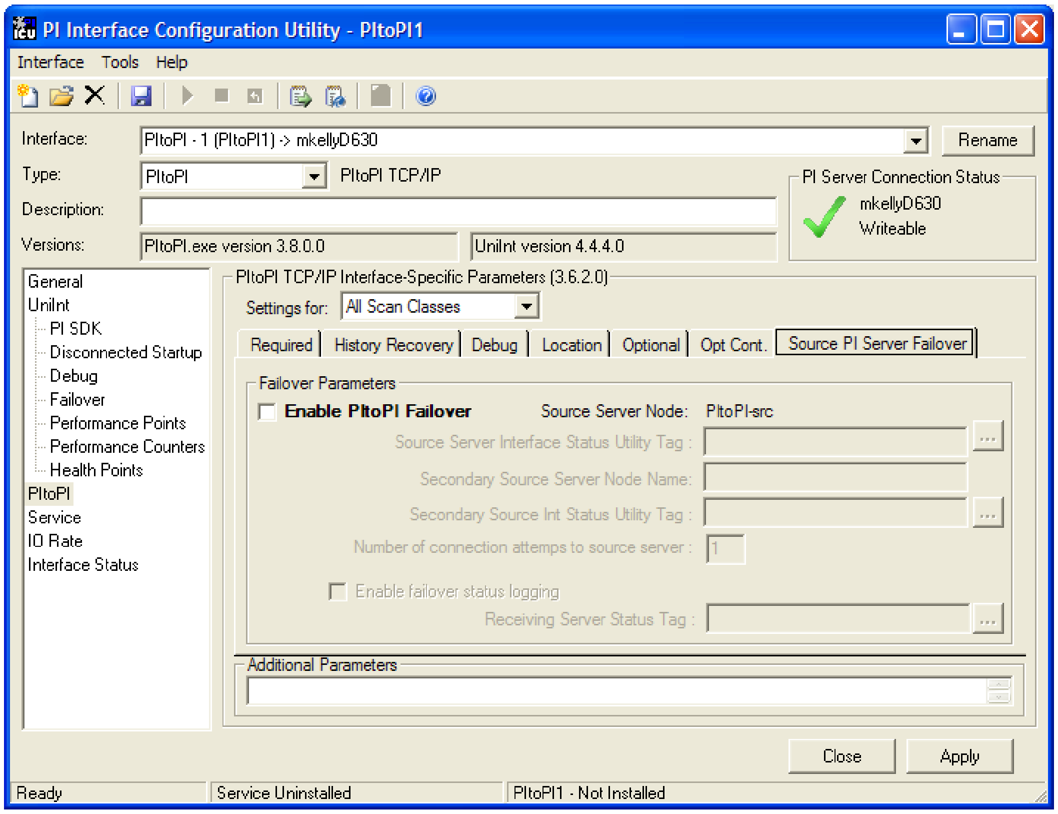
Receiving Host reconnection delay.

This parameter set the time delay for attempting to reconnect to receiving Historian Server after a disruption. The number is entered in seconds and converted to milliseconds before being saved in the batch file. This number must be between 1 second and 8 hours. (**`/DELAYR= x, default: 0 seconds`**)

Suppress writing I/O Timeout to tags upon reestablishment of a lost connection to the source Historian Server

When setting `Location3` to write "I/O Timeout" for any tags, use this parameter to suppress the I/O Timeout state written to these tags upon reestablishment of a lost connection to the source Historian Server. If this parameter is not set, the state written at reconnection will prevent history from being recovered for the period of the disconnection. (**`/TS`**)

Source Historian Server Failover Tab


PI Interface Configuration Utility - PItoPI1

Interface: PItoPI - 1 (PItoPI1) -> mkellyD630

Type: PItoPI PItoPI TCP/IP

Description:

Versions: PItoPI.exe version 3.8.0.0 UniInt version 4.4.4.0

PI Server Connection Status: mkellyD630 Writeable

PItoPI TCP/IP Interface-Specific Parameters (3.6.2.0)

Settings for: All Scan Classes

Failover Parameters

Enable PItoPI Failover Source Server Node: PItoPI-src

Source Server Interface Status Utility Tag :

Secondary Source Server Node Name:

Secondary Source Int Status Utility Tag :

Number of connection attemps to source server : 1

Enable failover status logging

Receiving Server Status Tag :

Additional Parameters

Ready | Service Uninstalled | PItoPI1 - Not Installed

Enable PItoPI Failover

This check box is used to allow the configuration of the failover. Until this box is checked none of the items on this tab are enabled. Note that having this check makes items 2 and 3 required since they are in yellow.

Source Server Interface Status Utility Tag

This is the name of a Historian Interface Status Utility tag configured on the source server defined in **`/SRC_HOST=`***`hostname`*. Click the *Browse* button to invoke the Tag Search utility to browse for this tag. (**`/SSU1=`***`tagname`*)

Secondary Source Server Node Name

This is the name of the second source node from which to retrieve data. This must be a Historian 3.x Historian Server node because the port number of 5450 will be appended to the end of this name when it is saved in the batch file. (**`/SEC_SRC=`***`nodename`***`:5450`**)

Secondary Source Int Status Utility Tag

This the name of a Historian Interface Status Utility tag configured on the source server defined in **`/SEC_SRC=`***`hostname`*. Click the *Browse* button to invoke the Tag Search utility to browse for this tag. (**`/SSU2=`***`tagname`*)

Number of connection attempts to source server

This parameter is used to specify the number of times to try connecting to source server if connection fails the first time. It can be used to restrict failover from occurring until after a certain number of attempts made to connect to the primary server have failed. The default number of attempts is 1.(**`/NT=x`**)

Enable failover status logging.

This check box is used to enable failover status logging. By checking this box, the user is allowed to enter a *Receiving Server Status Tag*. This `tagname` entered in this text box is the failover status tag configured on receiving server. Click the *Browse* button to invoke the Tag Search utility to browse for this tag. (**`/FST=`***`tagname`*)

Note: The *UniInt Interface User Manual* includes details about other command-line parameters, which may be useful.

Configuring Interface Startup Files

The interface has two startup configuration files; `PItoPI.bat` and `PItoPI.ini`. The `.bat` file is required and is the primary file for specifying interface configurations. The `.ini` file should only be used if configuring one copy of the interface to collect data from multiple source Historian Servers. In this configuration each scan class can be configured for a unique source Historian Server. This is not recommended for exception data collection. Interface performance is maximized by running a separate copy of the interface for each source Historian Server.

When using the `.ini` file, global parameters such as point source are defined in the `.bat` file. Scan class specific parameters are defined in the `.ini` file, such as source Historian Server. When a parameter is set in both the `.bat` and `.ini` file, the `.ini` file takes precedence.

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). The number of parameters is unlimited, and the maximum length of each parameter is 1024 characters.

Command-line Parameters

These parameters are displayed in five groups: General Interface Operation, History Recovery and Archive Data Collection, Exception Data Collection, Tag Attribute Override, and Server-level Failover.

General Interface Operation

.BAT	.INI	Description
`/db=#` Optional	DebugFlags	**`/db=1`** : Max debug **`/db=2`** : Startup processing **`/db=3`** : Historian Server connections **`/db=4`** : PI2 security validation **`/db=5`** : Tag additions, edits, deletions **`/db=6`** : Data read & writes **`/db=7`** : Failover Example: **`/db=2,4,5`**

`/delayr=#`
Optional
Default:
`/delayr=0`
	--	Millisecond time delay between reconnection attempts to the receiving Historian Server. Units are in milliseconds. Valid values are between 0 and 28800000ms (8 hours).

`/delays=#`
Optional
Default:
`/delays=0`
	--	Millisecond time delay between reconnection attempts to the source Historian Server. Valid values are between 0 and 28800000ms (8 hours).

`/ec=#` Optional	EventCounter	The first instance of the **`/ec`** parameter on the command-line is used to specify a counter number, **`#`**, for an I/O Rate point. Range allowed is 1-34 and 51-200. If the **`#`** is not specified, then the default event counter is 1. Also, if the **`/ec`** parameter is not specified at all, there is still a default event counter of 1 associated with the interface. If there is an I/O Rate point that is associated with an event counter of 1, each copy of the interface that is running without **`/ec=`**#explicitly defined will write to the same I/O Rate point. This means either explicitly defining an event counter other than 1 for each copy of the interface or not associating any I/O Rate points with event counter 1. Configuration of I/O Rate points is discussed in the section called I/O Rate Point.

`/f=SS.##`
or
`/f=SS.##,SS.##`
or
`/f=HH:MM:SS.##`
or
`/f=HH:MM:SS.##,`
`hh:mm:ss.##`
 	
Required
	--	The **`/f`** parameter defines the time period between scans in terms of hours (**`HH`**), minutes (**`MM`**), seconds (**`SS`**) and sub-seconds (**`##`**). The scans can be scheduled to occur at discrete moments in time with an optional time offset specified in terms of hours (**`hh`**), minutes (**`mm`**), seconds (**`ss`**) and sub-seconds (**`##`**). If **`HH`** and **`MM`** are omitted, then the time period that is specified is assumed to be in seconds.
		Each instance of the **`/f`** parameter on the command-line defines a scan class for the interface. There is no limit to the number of scan classes that can be defined. The first occurrence of the **`/f`** parameter on the command-line defines the first scan class of

.BAT	.INI	Description
		the interface; the second occurrence defines the second scan class, and so on. Historian Points are associated with a particular scan class via the Location4 Historian Point attribute. For example, all Historian Points that have Location4 set to 1 will receive input values at the frequency defined by the first scan class. Similarly, all points that have Location4 set to 2 will receive input values at the frequency specified by the second scan class, and so on. Two scan classes are defined in the following example: **`/f=00:01:00,00:00:05`** **`/f=00:00:07`** or, equivalently: **`/f=60,5 /f=7`** The first scan class has a scanning frequency of 1 minute with an offset of 5 seconds, and the second scan class has a scanning frequency of 7 seconds. When an offset is specified, the scans occur at discrete moments in time according to the formula: scan times = (reference time) + n(frequency) + offset where n is an integer and the reference time is midnight on the day that the interface was started. In the above example, frequency is 60 seconds and offset is 5 seconds for the first scan class. This means that if the interface was started at 05:06:06, the first scan would be at 05:07:05, the second scan would be at 05:08:05, and so on. Since no offset is specified for the second scan class, the absolute scan times are undefined. The definition of a scan class does not guarantee that the associated points will be scanned at the given frequency. If the interface is under a large load, then some scans may occur late or be skipped entirely. See the section “Performance Summaries” in the UniInt Interface User Manual.doc for more information on skipped or missed scans. **`Sub-second Scan Classes`** Sub-second scan classes can be defined on the command-line, such as **`/f=0.5 /f=00:00:00.1`** where the scanning frequency associated with the first scan class is 0.5 seconds and the scanning frequency associated with the second scan class is 0.1 of a second. Similarly, sub-second scan classes with sub-second offsets can be defined, such as **`/f=0.5,0.2 /f=1,0`** **`Wall Clock Scheduling`** Scan classes that strictly adhere to wall clock scheduling are now possible. This feature is available for interfaces that run on Windows.

.BAT	.INI	Description
		Previously, wall clock scheduling was possible, but not across daylight saving time. For example**,** `/f=24:00:00,08:00:00` corresponds to 1 scan a day starting at 8 AM. However, after a Daylight Saving Time change, the scan would occur either at 7 AM or 9 AM, depending upon the direction of the time shift. To schedule a scan once a day at 8 AM (even across daylight saving time), use `/f=24:00:00,00:08:00,L`. The `,L` at the end of the scan class tells UniInt to use the new wall clock scheduling algorithm.

`/host=name:port`
Required
	--	Name or IP address of receiving Historian Server.
		`Name` is the IP address of the Historian Sever node or the domain name of the Historian Server node. **`Port`** is the port number for TCP/IP communication. The port number is 545 (Historian 2) or 5450 (PI3). 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 an interface node, the domain name of the Historian 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. UniInt concatenates this string to the header that is used to identify error messages as belonging to a particular interface. See the Appendix A: Error and Informational Messages for more information. UniInt always uses the **`/id`** parameter in the fashion described above. This interface also 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`**

.BAT	.INI	Description
`/ist=tagname` Optional	--	Name of interface status tag. **`/ist=<tagname>`** where < **`tagname`**> is a digital tag on the receiving Historian Server.
`/ln=username` Optional	SourceLogin	Login name of PI user on Historian 2 node. This is used when source server is Historian 2.

`/oc=#`
Optional
Default:
`/oc=30`
	--	Number of seconds between calculating time offset between the interface and Historian Server nodes.

`/ps=x` Required	--	The **`/ps`** parameter specifies the point source for the interface. **`X`** is not case sensitive and can be any single or 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 Historian Points. The interface will attempt to load only those Historian Points with the appropriate point source.
`/pw=password` Optional	SourcePassword	Login password of PI user on Historian 2 node. This is used when source server is Historian 2.
`/sf=filename` Required for Historian 2 source	SecurityFile	Used for locating the security file on a PI2 source server. This switch specifies the <name> part of the file name. Note that the complete file name must have this format: **`PItoPI<name>.SEC`** where <name> is the portion specified by **`/sf`**
`/src_host=name:port` Required	SourceHost	Name or IP address of source Historian Server. **`/src_host=node_name:tcpip_port`** The port number is 545 (Historian 2) or 5450 (Historian 3).

.BAT	.INI	Description

`/stopstat`
or
`/stopatat=`
`digstate`
Default:
`/stopstat=`
`"Intf shut"`
Optional
	--	If the **`/stopstat`** parameter is present on the startup command line, then the digital state **`Intf Shut`** will be written to each Historian Point when the interface is stopped.
		If **`/stopstat=digstate`** is present on the command line, then the digital state, **`digstate`**, will be written to each Historian Point when the interface is stopped. UniInt uses the first occurrence in the table.
		If neither **`/stopstat`** nor **`/stopstat=digstate`** is specified on the command line, then no digital states will be written when the interface is shut down.
		Note: The **`/stopstat`** parameter is disabled If the interface is running in a UniInt failover configuration as defined in the UniInt Failover Configuration section of this manual. Therefore, the digital state, **`digstate`**, will not be written to each Historian Point when the interface is stopped. This prevents the digital state being written to Historian Points while a redundant system is also writing data to the same Historian Points. The **`/stopstat`** parameter is disabled even if there is only one interface active in the failover configuration.
		Examples:
		`/stopstat=shutdown`
		`/stopstat="Intf Shut"`
		The entire **`digstate`** value should be enclosed within double quotes when there is a space in **`digstate`**.

`/ts` Optional	--	Suppress 'IO Timeout' events when reconnecting to source Historian Server. These events are configured through **`Location3`** attribute. If this switch is not set, the event written at reconnection will prevent history from being recovered for the period of the disconnection.

History Recovery and Archive Data Collection

.BAT	.INI	Description
`/dc` Optional	--	Apply data compression to history recovery and archive scan updates. The default behavior is for this data to bypass compression. This switch must be specified to prevent data mismatches if tags are configured to include snapshot value with archive scan updates.
`/hronly= start,end` Optional	HistOnly	Used without "=*start*,*end*" to disable exception data collection. Also used to specify time range specific history recovery: **`/hronly=starttime,endtime`** The times must be specified using Historian Time string formats with a colon or underscore separating the date and the time: **`/hronly=dd-mmm-yy:hh:mm:ss,dd-mmm-yy:hh:mm:ss`** or **`/hronly=dd-mmm-yy_hh:mm:ss,dd-mmm-yy_hh:mm:ss`** For example: **`/hronly=10-dec-98:10:00,10-dec-98:12:00`** or **`/hronly=10-dec-98_10:00,10-dec-98_12:00`** **`Note: timestamps are local to where the interface runs. This is important if source/receiving Historian Server are in a different timezone.`** When configured for time range specific history recovery the interface recovers data then exits.

`/hrpause=#`
Optional
Default:
`/hrpause=0`
	HistPause	Milliseconds to pause between tags during history recovery. Used to throttle archive data retrieval during history recovery.

`/mh=x`
Optional
Default:
`/mh=1000`
	--	Available for PI 3.3 or later receiving Historian Servers.
		Sets the maximum number of archive events retrieved per data request. During history recovery and archive data collection, the interface specifies the maximum number of events to return. If more than the maximum exist, the interface makes multiple calls until all events are retrieved for the time period.
		Increasing the default may increase data throughput for archive data retrieval.

.BAT	.INI	Description
/ns Optional	--	Boundary condition for data retrieval with a time-range specific history recovery. When specified the interface will start history recovery beginning with the first value prior to the start time. The default behavior is to begin with the first value after the start time.
/rh=# Optional Default: **/rh=8**	HistRecovery	Hours of history recovery to perform. There is no limit on the number of hours for history recovery. However special preparations may be necessary if the recovery period spans beyond the primary archive. For example there may not be enough space in the target (non-primary) archive for the recovery data, or non-primary archives may not have space allocated for newly created tags. See the Historian Server System Management Guide for information on backfilling data.
/rh_inc=# Optional Default: **/rh_inc=24**	MaxArcTimespan	Time increments within the total **/rh** recovery period. For example, **/rh**=48 and **/rh_inc**=24. The interface will cycle through the tag list twice. On the first cycle, data recovery is performed for the first 24-hour period. On the second cycle, the second 24-hour period is collected to complete the total 48 hour recovery period.
/rh_qcheck=# Optional Default: **/rh_qcheck=5**		For exception data scan classes, sets the time interval between clearing the exception queue on the source Historian Server during history recovery. By default the interface will collect exceptions from the source Historian Server every 5 seconds during history recovery to prevent overflowing the queue. Users may want to adjust this time interval to tune history recovery performance.

Exception Data Collection

.BAT	.INI	Description
/me=# Optional Default: **/me=5000**	--	Sets the maximum number of exception events retrieved per data request. A large count reduces the number of calls required for acquiring exception updates. A small count reduces the time to complete each request (for troubleshooting network timeout issues).
/sn Optional	--	Bypass exception filtering for data collected from the source Historian Server. This data has already passed exception for the source tag so additional data filtering can only lead to data mismatches between Historian Servers.

Tag Attribute Override

.BAT	.INI	Description
`/c1` Optional	--	**`Location1`** tag attribute override. Load all tags configured for the specified point source regardless of Location1and interface ID values.
`/c2=#` Optional	--	**`Location2`** tag attribute override. Ignore **`Location2`** for each tag and use the specified value. Valid values are 0-7.
`/c3=#` Optional	--	**`Location3`** tag attribute override. Ignore **`Location3`** for each tag and use the specified value. Valid values are 0-15.
`/c4=#` Optional	--	**`Location4`** tag attribute override. Ignore **`Location4`** for each tag and use the specified value. Valid values are 1 (exception data collection) or 2 (archive data collection).
`/c5=#` Optional	--	**`Location5`** tag attribute override. Ignore **`Location5`** for each tag and use the specified value. Valid values are 0-3.
`/ptid` Optional	--	Source tag point ID is specified in **`UserInt1`** attribute. Ignore **`InstrumentTag`**, **`Exdesc`** and Tag **`name`** attributes. Note: this switch is not compatible with source Historian Server failover since point IDs will not necessarily match between source Historian Servers. If the source Historian Servers are part of a Historian Collective use **`/tn`** instead.
`/tn` Optional	--	Source tag name is same as interface tag name. Ignore **`InstrumentTag`**, **`Exdesc`** and **`UserInt1`** attributes.
`/tnex` Optional	--	Source tag name is located in the **`Exdesc`** or Tag name attribute and the **`InstrumentTag`** and **`UserInt1`** attributes are ignored for identifying source tag.

Server-Level Failover

.BAT	.INI	Description
`/fst=tag` Optional	--	Name of failover status tag. **`/fst=<tagname>`** where <tagname> is a digital tag on the receiving Historian Server.

`/nt=#`
Optional
Default:
`/nt=1`
	--	Number of reconnection attempts to source Historian Server before initiating a failover. Valid values are 0 and greater.
		Prevents failover flip-flop when experiencing intermittent network updates.

.BAT	.INI	Description
/sec_src= node :port Required	--	Name or IP address of the secondary source Historian Server. **/sec_src=node_name:tcpip_port** The port number is 5450 (PI3). PI2 is not supported for source Historian Server failover.
/ssu1=tag Required	--	Historian Interface Status Utility tag name for the **/src_host** source Historian Server. **/ssu1=<tagname>** Required for monitoring source data quality (current or stale data).
/ssu2=tag Required	--	Historian Interface Status Utility tag name for the **/sec_src** source Historian Server. **/ssu2=<tagname>** Required for monitoring source data quality (current or stale data).

Sample Startup Configuration Files

The startup files for the interface reside in the directory `PIHOME\Interfaces\PItoPI` where `PIHOME` is defined in `%WINDIR%\pipc.ini` by the installation program. Typically, `PIHOME` is `c:\pipc`. The startup files consist of `PItoPI.bat` and `PItoPI.ini`.

Sample PItoPI.bat File

```
REM=================================================================
REM
REM PItoPI.bat
REM
REM Sample startup file for the PItoPI TCP/IP Interface
REM
REM=================================================================
REM
REM Rockwell Automation strongly recommends using ICU to modify startup
files.
REM
REM  Sample command line
REM
        .\PItoPI.exe ^
        /host=XXXXXX:5450 ^
        /src_host=XXXXXX:5450 ^
        /ps=PItoPI ^
        /id=1 ^
        /f=10

REM End of PItoPI.bat File
```

Sample PItoPI.ini File

```
;                               Sample PItoPI.ini
;
;-------------------------------------------------------------------
; Purpose:
;       This file should be used in conjunction with PItoPI.bat. It is only
;       required when a user wishes to collect data from multiple source
;       servers with a single copy of the interface.
;
;       the headings read [FT-PItoPI-x.y] where;
;       x = interface id
;       y = scan class (if specified)
;-------------------------------------------------------------------
;
[FT-PItoPI-1]
;EventCounter=1
;MaxArcTimespan=24
;
[FT-PItoPI-1.1]
;SourceHost=XXXXXX:5450
;HistRecovery=48
;
[FT-PItoPI-1.2]
;SourceHost=XXXXXX:5450
;HistRecovery=72
;
;--------------------------------------------------------------------
; List of possible parameters
;
;SourceHost=XXXXXX:5450      Name of source Historian Server,
;                            port=5450 for PI3 and 545 for PI2. This
;                            field MUST BE an IP address.
;ReceivingHost=XXXXXX:5450   Name of the receiving Historian Server,
;                            port=5450 for PI3 and 545 for PI2. This
;                            field MUST BE an IP address.
;SecurityFile=securityfile   Required if PI2, <name> part of security file
;                            PItoPI<name>.SEC
;SourceLogin=userid          PI2 PI user name
;SourcePassword=password     PI2 PI user password
;EventCounter=#              Number of event counter defined in
;                            \dat\iorates.dat file
;HistRecovery=#              total hours of history recovery, default=8hrs
;MaxArcTimespan=#            history recovery increment, divided into total
;                            hours of history recovery (HistRecovery),
;                            default=24hrs
;HistPause=#                 pause between history recovery increments in
;                            milliseconds
;HistOnly=#                  flag to disable exception data collection
;                            (0=off, 1=on)
;DebugFlags=#,#,#,#...       Generates additional messages for
troubleshooting
;                            comma separated list: 1,2,3,4,5,6,7
;--------------------------------------------------------------------
; end of sample PItoPI.ini
```

Chapter 9. UniInt Failover Configuration

Introduction

To minimize data loss during a single point of failure within a system, UniInt provides two failover schemas: (1) synchronization through the data source and (2) synchronization through a shared file. Synchronization through the data source is *Phase 1*, and synchronization through a shared file is *Phase 2*.

Phase 1 UniInt Failover uses the data source itself to synchronize failover operations and provides a *hot failover*, *no data loss* solution when a single point of failure occurs. For this option, the data source must be able to communicate with and provide data for two interfaces simultaneously. Additionally, the failover configuration requires the interface to support outputs.

Phase 2 UniInt Failover uses a shared file to synchronize failover operations and provides for *hot*, *warm*, or *cold failover*. The Phase 2 hot failover configuration provides a *no data loss* solution for a single point of failure similar to Phase 1. However, in warm and cold failover configurations, you can expect a small period of data loss during a single point of failure transition.

Note: This interface supports only Phase 2 failover.

You can also configure the UniInt interface level failover to send data to a High Availability (HA) Historian Server collective. The collective provides redundant Historian Servers to allow for the uninterrupted collection and presentation of Historian Time series data. In an HA configuration, Historian Servers can be taken down for maintenance or repair. The HA Historian Server collective is described in the *Historian Server Reference Guide*.

When configured for UniInt failover, the interface routes all FactoryTalk Historian data through a state machine. The state machine determines whether to queue data or send it directly to Historian depending on the current state of the interface. When the interface is in the active state, data sent through the interface gets routed directly to Historian. In the backup state, data from the interface gets queued for a short period. Queued data in the backup interface ensures a *no-data loss* failover under normal circumstances for Phase 1 and for the hot failover configuration of Phase 2. The same algorithm of queuing events while in backup is used for output data.

Quick Overview

The Quick Overview below may be used to configure this Interface for failover. The failover configuration requires the two copies of the interface participating in failover be installed on different nodes. Users should verify non-failover interface operation as discussed in the Installation Checklist section of this manual prior to configuring the interface for failover operations. If you are not familiar with UniInt failover configuration, return to this section after reading the rest of the UniInt Failover Configuration section in detail. If a failure occurs at any step below, correct the error and start again at the beginning of step 6 Test in the table below. For the discussion below, the first copy of the interface configured and tested will be considered the primary interface and the second copy of the interface configured will be the backup interface.

Configuration

- One Data Source
- Two Interfaces

Prerequisites

- Interface 1 is the Primary interface for collection of FactoryTalk Historian data from the data source.
- Interface 2 is the Backup interface for collection of FactoryTalk Historian data from the data source.
- You must set up a shared file.
- Phase 2: The shared file must store data for five failover tags: (1) Active ID, (2) Heartbeat 1, (3) Heartbeat 2, (4) Device Status 1 and (5) Device Status 2.
- Each interface must be configured with two required failover command line parameters: (1) its FailoverID number (**`/UFO_ID`**); (2) the FailoverID number of its Backup interface (**`/UFO_OtherID`**). You must also specify the name of the Historian Server host for exceptions and Historian tag updates.
- All other configuration parameters for the two interfaces must be identical.

Synchronization through a Shared File (Phase 2)

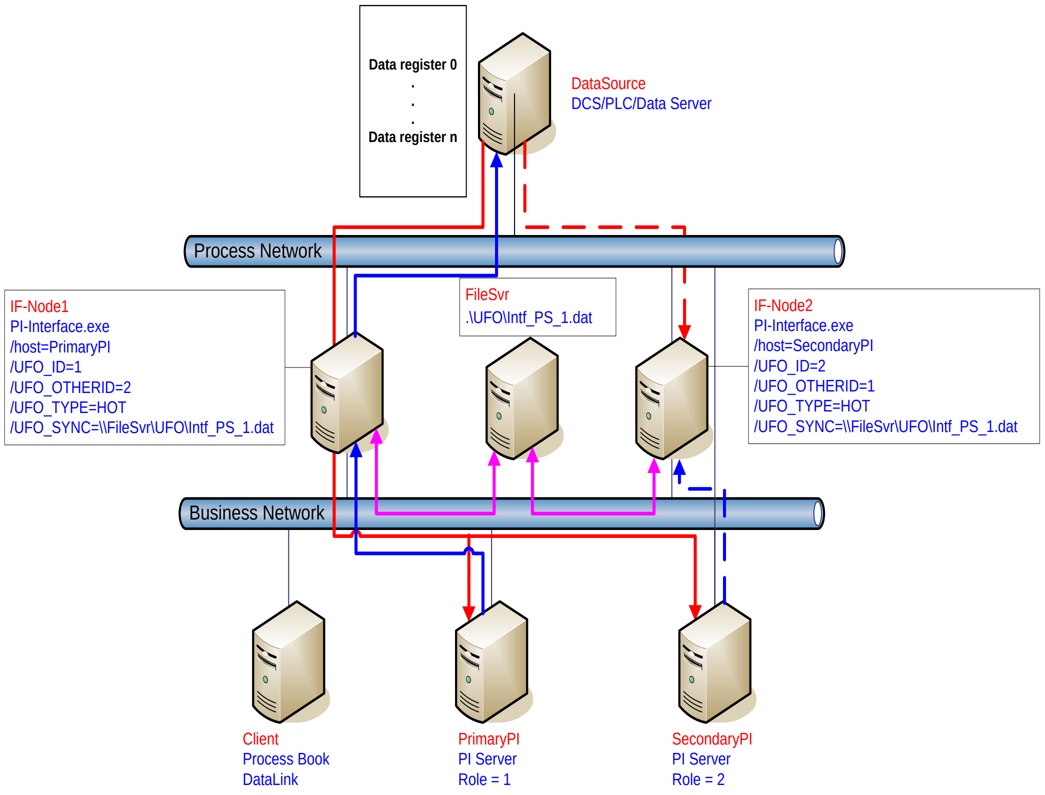


Figure 4: Synchronization through a Shared File (Phase 2) Failover Architecture

The Phase 2 failover architecture is shown in Figure 4 which depicts a typical network setup including the path to the synchronization file located on a File Server (FileSvr). Other configurations may be supported and this figure is used only as an example for the following discussion.

For a more detailed explanation of this synchronization method, see Detailed Explanation of Synchronization through a Shared File (Phase 2)

Configuring Synchronization through a Shared File (Phase 2)

Step	Description
1.	Verify non-failover interface operation as described in the Installation Checklist section of this manual
2.	**Configure the Shared File** Choose a location for the shared file. The file can reside on one of the interface nodes but Rockwell Automation strongly recommends that you put the file on a dedicated file server that has no other role in data collection. Setup a file share and make sure to assign the permissions so that both Primary and Backup interfaces have read/write access to the file.
3.	**Configure the interface parameters** Use the Failover section of the Interface Configuration Utility (ICU) to enable failover and create two parameters for each interface: (1) a Failover ID number for the interface; and (2) the Failover ID number for its backup interface. The Failover ID for each interface must be unique and each interface must know the Failover ID of its backup interface. If the interface can perform using either Phase 1 or Phase 2 pick the Phase 2 radio button in the ICU. Select the synchronization File Path and File to use for Failover. Select the type of failover required (Cold, Warm, Hot). The choice depends on what types of failover the interface supports. Ensure that the user name assigned in the “Log on as:” parameter in the Service section of the ICU is a user that has read/write access to the folder where the shared file will reside. All other command line parameters for the primary and secondary interfaces must be identical. If you use a Historian Collective, you must point the primary and secondary interfaces to different members of the collective by setting the SDK Member under the PI Host Information section of the ICU. [Option] Set the update rate for the heartbeat point if you need a value other than the default of 5000 milliseconds.
4.	**Configure the Historian tags** Configure five Historian tags for the interface: the Active ID, Heartbeat 1, Heartbeat2, Device Status 1 and Device Status 2. You can also configure two state tags for monitoring the status of the interfaces. Do not confuse the failover Device status tags with the UniInt Health Device Status tags. The information in the two tags is similar, but the failover device status tags are integer values and the health device status tags are string values.

Tag	ExDesc	digitalset	
ActiveID	[UFO2_ACTIVEID]		UniInt does not examine the remaining attributes, but the PointSource and Location1 must match
IF1_Heartbeat (IF-Node1)	[UFO2_HEARTBEAT:#]		
IF2_Heartbeat (IF-Node2)	[UFO2_HEARTBEAT:#]		
IF1_DeviceStatus (IF-Node1)	[UFO2_DEVICESTAT:#]		
IF2_DeviceStatus (IF-Node2)	[UFO2_DEVICESTAT:#]		
IF1_State (IF-Node1)	[UFO2_STATE:#]	IF_State	
IF2_State (IF-Node2)	[UFO2_STATE:#]	IF_State	

Step	Description
5.	**Test the configuration.** After configuring the shared file and the interface and Historian tags, the interface should be ready to run. See Troubleshooting UniInt Failover for help resolving Failover issues. 1. Start the primary interface interactively without buffering. 2. Verify a successful interface start by reviewing the `pipc.log` file. The log file will contain messages that indicate the failover state of the interface. A successful start with only a single interface copy running will be indicated by an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface not available.`" If the interface has failed to start, an error message will appear in the log file. For details relating to informational and error messages, refer to the Messages section below. 3. Verify data on the Historian Server using available Historian tools. • The **Active ID** control tag on the Historian Server must be set to the value of the running copy of the interface as defined by the **`/UFO_ID`** startup command-line parameter. • The **Heartbeat** control tag on the Historian Server must be changing values at a rate specified by the **`/UFO_Interval`** startup command-line parameter. 4. Stop the primary interface. 5. Start the backup interface interactively without buffering. Notice that this copy will become the primary because the other copy is stopped. 6. Repeat steps 2, 3, and 4. 7. Stop the backup interface. 8. Start buffering. 9. Start the primary interface interactively. 10. Once the primary interface has successfully started and is collecting data, start the backup interface interactively. 11. Verify that both copies of the interface are running in a failover configuration. • Review the `pipc.log` file for the copy of the interface that was started first. The log file will contain messages that indicate the failover state of the interface. The state of this interface must have changed as indicated with an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface available.`" If the interface has not changed to this state, browse the log file for error messages. For details relating to informational and error messages, refer to the Messages section below. • Review the `pipc.log` file for the copy of the interface that was started last. The log file will contain messages that indicate the failover state of the interface. A successful start of the interface will be indicated by an informational message stating "`UniInt failover: Interface in the "Backup" state.`" If the interface has failed to start, an error message will appear in the log file. For details relating to informational and error messages, refer to the Messages section below. 12. Verify data on the Historian Server using available Historian tools. • The **Active ID** control tag on the Historian Server must be set to the value of the running copy of the interface that was started first as

Step	Description
	defined by the **`/UFO_ID`** startup command-line parameter. • The **Heartbeat** control tags for both copies of the interface on the Historian Server must be changing values at a rate specified by the **`/UFO_Interval`** startup command-line parameter or the scan class which the points have been built against. 13. Test Failover by stopping the primary interface. 14. Verify the backup interface has assumed the role of primary by searching the `pipc.log` file for a message indicating the backup interface has changed to the "`UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface not available`." The backup interface is now considered primary and the previous primary interface is now backup. 15. Verify no loss of data in Historian. There may be an overlap of data due to the queuing of data. However, there must be no data loss. 16. Start the backup interface. Once the primary interface detects a backup interface, the primary interface will now change state indicating "`UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface available.`" In the `pipc.log` file. 17. Verify the backup interface starts and assumes the role of backup. A successful start of the backup interface will be indicated by an informational message stating "`UniInt failover: Interface in "Backup state.`" Since this is the initial state of the interface, the informational message will be near the beginning of the start sequence of the `pipc.log` file. 18. Test failover with different failure scenarios (e.g. loss of Historian connection for a single interface copy). UniInt failover guarantees no data loss with a single point of failure. Verify no data loss by checking the data in Historian and on the data source. 19. Stop both copies of the interface, start buffering, start each interface as a service. 20. Verify data as stated above. 21. To designate a specific interface as primary. Set the **Active ID** point on the Data Source Server of the desired primary interface as defined by the **`/UFO_ID`** startup command-line parameter.

Configuring UniInt Failover through a Shared File (Phase 2)

Start-Up Parameters

Note: The **`/stopstat`** parameter is disabled If the interface is running in a UniInt failover configuration. Therefore, the digital state, `digstate`, will not be written to each Historian Point when the interface is stopped. This prevents the digital state being written to Historian Points while a redundant system is also writing data to the same Historian Points. The **`/stopstat`** parameter is disabled even if there is only one interface active in the failover configuration.

The following table lists the start-up parameters used by UniInt Failover Phase 2. All of the parameters are required except the **`/UFO_Interval`** startup parameter. See the table below for further explanation.

Parameter	Required/ Optional	Description	Value/Default
`/UFO_ID=#`	Required	Failover ID for IF-Node1 This value must be different from the failover ID of IF-Node2.	Any positive, non-zero integer / **1**
	Required	Failover ID for IF-Node2 This value must be different from the failover ID of IF-Node1.	Any positive, non-zero integer / **2**
`/UFO_OtherID=#`	Required	Other Failover ID for IF-Node1 The value must be equal to the Failover ID configured for the interface on IF-Node2.	Same value as Failover ID for IF-Node2 / **2**
	Required	Other Failover ID for IF-Node2 The value must be equal to the Failover ID configured for the interface on IF-Node1.	Same value as Failover ID for IF-Node1 / **1**

`/UFO_Sync=`
`path/[filename]`
	Required for Phase 2 synchronization	The Failover File Synchronization Filepath and Optional Filename specify the path to the shared file used for failover synchronization and an optional filename used to specify a user defined filename in lieu of the default filename.
		The *path* to the shared file directory can be a fully qualified machine name and directory, a mapped drive letter, or a local path if the shared file is on one of the interface nodes. The *path* must be terminated by a slash (/) or backslash (\ ) character. If no terminating slash is found, in the **`/UFO_Sync`** parameter, the interface interprets the final character string as an optional *filename*.
		The optional *filename* can be any valid filename. If the file does not
			Any valid pathname / any valid filename
			The default filename is generated as *executablename_ pointsource_ interfaceID.dat*

Parameter	Required/ Optional	Description	Value/Default
		exist, the first interface to start attempts to create the file. **Note:** If using the optional filename, **do not** supply a terminating slash or backslash character. If there are any spaces in the *path* or *filename*, the entire path and filename must be enclosed in quotes. **Note:** If you use the backslash and path separators and enclose the path in double quotes, the final backslash must be a double backslash (`\\`). Otherwise the closing double quote becomes part of the parameter instead of a parameter separator. Each node in the failover configuration must specify the same path and filename and must have read, write, and file creation rights to the shared directory specified by the *path* parameter. The service that the interface runs against must specify a valid logon user account under the "Log On" tab for the service properties.	
`/UFO_Type=type`	Required	The Failover Type indicates which type of failover configuration the interface will run. The valid types for failover are HOT, WARM, and COLD configurations. If an interface does not supported the requested type of failover, the interface will shutdown and log an error to the `pipc.log` file stating the requested failover type is not supported.	COLD|WARM|HOT / **COLD**
`/UFO_Interval=#`	Optional	Failover Update Interval Specifies the heartbeat Update Interval in milliseconds and must be the same on both interface computers. This is the rate at which UniInt updates the Failover Heartbeat tags as well as how often UniInt checks on the status of the other copy of the interface.	50 - 20000 / **1000**

Parameter	Required/ Optional	Description	Value/Default
`/Host=server`	Required	Host Historian Server for Exceptions and Historian tag updates The value of the **`/Host`** startup parameter depends on the Historian Server configuration. If the Historian Server is not part of a collective, the value of **`/Host`** must be identical on both interface computers. If the redundant interfaces are being configured to send data to a Historian Server collective, the value of the **`/Host`** parameters on the different interface nodes should equal to different members of the collective. This parameter ensures that outputs continue to be sent to the Data Source if one of the Historian Servers becomes unavailable for any reason.	For IF-Node1 PrimaryPI / **None** For IF-Node2 SecondaryPI / **None**

Failover Control Points

The following table describes the points that are required to manage failover. In Phase 2 Failover, these points are located in a data file shared by the Primary and Backup interfaces.

Rockwell Automation recommends that you locate the shared file on a dedicated server that has no other role in data collection. This avoids potential resource contention and processing degradation if your system monitors a large number of data points at a high frequency.

Point	Description	Value / Default
ActiveID	Monitored by the interfaces to determine which interface is currently sending data to Historian. **ActiveID** must be initialized so that when the interfaces read it for the first time, it is not an error. **ActiveID** can also be used to force failover. For example, if the current Primary is IF-Node 1 and **ActiveID** is 1, you can manually change **ActiveID** to 2. This causes the interface at IF-Node2 to transition to the primary role and the interface at IF-Node1 to transition to the backup role.	From 0 to the highest Interface Failover ID number / **None**) Updated by the redundant Interfaces Can be changed manually to initiate a manual failover
Heartbeat 1	Updated periodically by the interface on IF-Node1. The interface on IF-Node2 monitors this value to determine if the interface on IF-Node1 has become unresponsive.	Values range between 0 and 31 / **None** Updated by the interface on IF-Node1
Heartbeat 2	Updated periodically by the interface on IF-Node2. The interface on IF-Node1 monitors this value to determine if the interface on IF-Node2 has become unresponsive.	Values range between 0 and 31 / **None** Updated by the interface on IF-Node2

Historian Tags

The following tables list the required UniInt Failover Control Historian tags, the values they will receive, and descriptions.

Active_ID Tag Configuration

Attributes	ActiveID
Tag	**<Intf>_ActiveID**
ExDesc	[UFO2_ActiveID]
Location1	Match # in **`/id=#`**
Location5	Optional, Time in minutes to wait for backup to collect data before failing over.
Point Source	Match x in **`/ps=x`**
Point Type	Int32
Shutdown	0
Step	1

Heartbeat and Device Status Tag Configuration

Attribute	Heartbeat 1	Heartbeat 2	DeviceStatus 1	DeviceStatus 2
Tag	<HB1>	<HB2>	<DS1>	<DS2>
ExDesc	[UFO2_Heartbeat:#] Match # in **`/UFO_ID=#`**	[UFO2_Heartbeat:#] Match # in **`/UFO_OtherID=#`**	[UFO2_DeviceStat:#] Match # in **`/UFO_ID=#`**	[UFO2_DeviceStat:#] Match # in **`/UFO_OtherID=#`**
Location1	Match # in **`/id=#`**	Match # in **`/id=#`**	Match # in **`/id=#`**	Match # in **`/id=#`**
Location5	Optional, Time in minutes to wait for backup to collect data before failing over.	Optional, Time in minutes to wait for backup to collect data before failing over.	Optional, Time in minutes to wait for backup to collect data before failing over.	Optional, Time in minutes to wait for backup to collect data before failing over.
Point Source	Match x in **`/ps=x`**	Match x in **`/ps=x`**	Match x in **`/ps=x`**	Match x in **`/ps=x`**
Point Type	int32	int32	int32	int32
Shutdown	0	0	0	0
Step	1	1	1	1

Interface State Tag Configuration

Attribute	Primary	Backup
Tag	<Tagname1>	<Tagname2>
DigitalSet	UFO_State	UFO_State
ExDesc	[UFO2_State:#] (Match **`/UFO_ID=#`** on primary node)	[UFO2_State:#] (Match **`/UFO_ID=#`** on backup node)
Location1	Match # in **`/id=#`**	Same as for Primary node
PointSource	Match x in **`/ps=x`**	Same as for Primary node
PointType	digital	digital
Shutdown	0	0
Step	1	1

The following table describes the extended descriptor for the above Historian tags in more detail.

Historian Tag ExDesc	**Required / Optional**	**Description**	**Value**
[UFO2_ACTIVEID]	Required	Active ID tag The ExDesc must start with the case sensitive string: [UFO2_ACTIVEID]. The PointSource must match the interfaces' point source. Location1 must match the ID for the interfaces. Location5 is the COLD failover retry interval in minutes. This can be used to specify how long before an interface retries to connect to the device in a COLD failover configuration. (See the description of COLD failover retry interval for a detailed explanation.)	0 - highest Interface Failover ID Updated by the redundant Interfaces
[UFO2_HEARTBEAT:#] (IF-Node1)	Required	Heartbeat 1 Tag The ExDesc must start with the case sensitive string: [UFO2_HEARTBEAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node1. The PointSource must match the interfaces' point source. Location1 must match the ID for the interfaces.	0 - 31 / **None** Updated by the interface on IF-Node1
[UFO2_HEARTBEAT:#] (IF-Node2)	Required	Heartbeat 2 Tag The ExDesc must start with the case sensitive string: [UFO2_HEARTBEAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2. The PointSource must match the interfaces' point source. Location1 must match the ID for the interfaces.	0 - 31 / **None** Updated by the interface on IF-Node2

Historian Tag ExDesc	Required / Optional	Description	Value
[UFO2_DEVICESTAT :#] (IF-Node1)	Required	Device Status 1 Tag The ExDesc must start with the case sensitive string: [UFO2_HEARTBEAT:#] The value following the colon (:) must be the Failover ID for the interface running on IF-Node1 The PointSource must match the interfaces' point source. Location1 must match the ID for the interfaces. A lower value is a better status and the interface with the lower status will attempt to become the primary interface. The failover 1 device status tag is very similar to the UniInt Health Device Status tag except the data written to this tag are integer values. A value of 0 is good and a value of 99 is OFF. Any value between these two extremes may result in a failover. The interface client code updates these values when the health device status tag is updated.	0 - 99 / **None** Updated by the interface on IF-Node1
[UFO2_DEVICESTAT :#] (IF-Node2)	Required	Device Status 2 Tag The ExDesc must start with the case sensitive string: [UFO2_HEARTBEAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2 The PointSource must match the interfaces' point source. Location1 must match the ID for the interfaces. A lower value is a better status and the interface with the lower status will attempt to become the primary interface.	0 - 99 / **None** Updated by the interface on IF-Node2
[UFO2_STATE:#] (IF-Node1)	Optional	State 1 Tag The ExDesc must start with the case sensitive string: [UFO2_STATE:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node1 The failover state tag is recommended. The failover state tags are digital tags assigned to a digital state set with the following values. 0 = Off: The interface has been shut down. 1 = Backup No Data Source: The	0 - 5 / **None** Normally updated by the interface currently in the primary role.

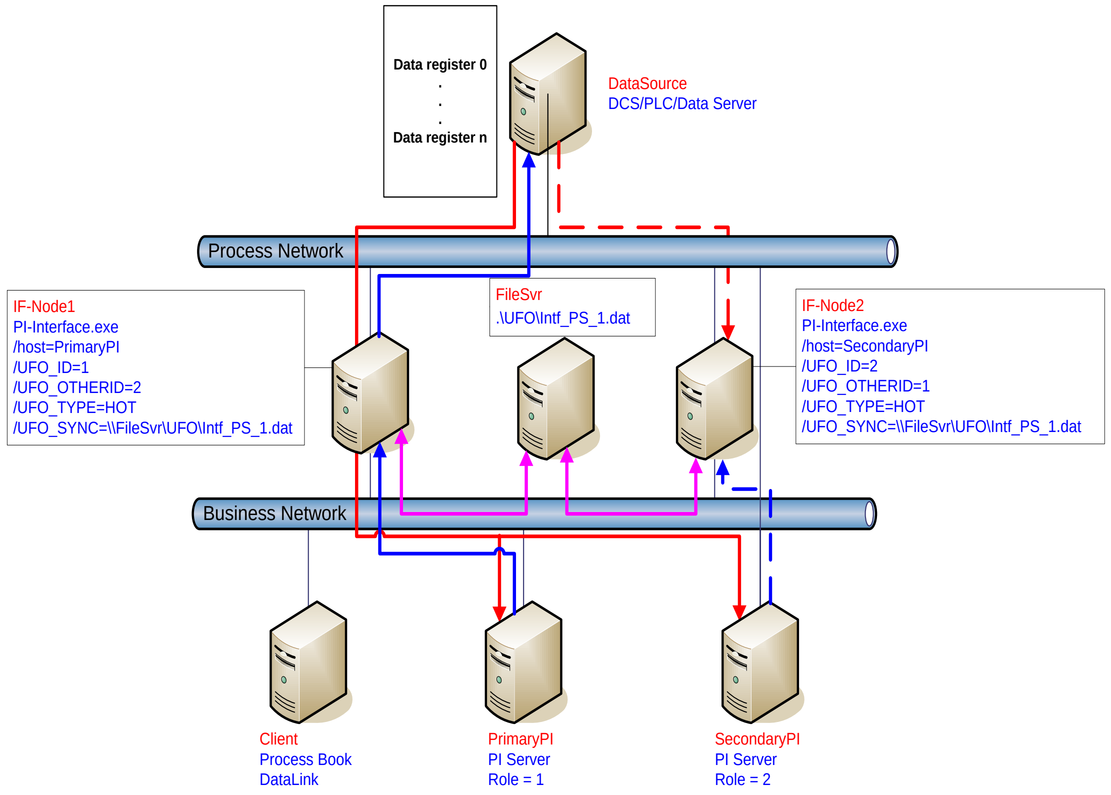
Historian Tag ExDesc	Required / Optional	Description	Value
		interface is running but cannot communicate with the data source. 2 = Backup No Historian Connection: The interface is running and connected to the data source but has lost its communication to the Historian Server. 3 = Backup: The interface is running and collecting data normally and is ready to take over as primary if the primary interface shuts down or experiences problems. 4 = Transition: The interface stays in this state for only a short period of time. The transition period prevents thrashing when more than one interface attempts to assume the role of primary interface. 5 = Primary: The interface is running, collecting data and sending the data to Historian.	
[UFO2_STATE:#] (IF-Node2)	Optional	State 2 Tag The ExDesc must start with the case sensitive string: [UFO2_STATE:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2 The failover state tag is recommended.	Normally updated by the interface currently in the Primary state. Values range between 0 and 5. See description of State 1 tag.

Detailed Explanation of Synchronization through a Shared File (Phase 2)

In a shared file failover configuration, there is no direct failover control information passed between the data source and the interface. This failover scheme uses five Historian tags to control failover operation, and all failover communication between primary and backup interfaces passes through a shared data file.

Once the interface is configured and running, the ability to read or write to the Historian tags is not required for the proper operation of failover. This solution does not require a connection to the Historian Server after initial startup because the control point data are set and monitored in the shared file. However, the Historian tag values are sent to the Historian Server so that you can monitor them with standard Rockwell Automation client tools.

You can force manual failover by changing the **ActiveID** on the data source to the backup failover ID.



The figure above shows a typical network setup in the normal or steady state. The solid magenta lines show the data path from the interface nodes to the shared file used for failover synchronization. The shared file can be located anywhere in the network as long as both interface nodes can read, write, and create the necessary file on the shared file machine. Rockwell Automation strongly recommends that you put the file on a dedicated file server that has no other role in the collection of data.

The major difference between synchronizing the interfaces through the data source (Phase 1) and synchronizing the interfaces through the shared file (Phase 2) is where the control data is located. When synchronizing through the data source, the control data is acquired directly from the data source. We assume that if the primary interface cannot read the failover control points, then it cannot read any other data. There is no need for a backup communications path between the control data and the interface.

When synchronizing through a shared file, however, we cannot assume that loss of control information from the shared file implies that the primary interface is down. We must account for the possible loss of the path to the shared file itself and provide an alternate control path to determine the status of the primary interface. For this reason, if the shared file is unreachable for any reason, the interfaces use the Historian Server as an alternate path to pass control data.

When the backup interface does not receive updates from the shared file, it cannot tell definitively why the primary is not updating the file, whether the path to the shared file is down, whether the path to the data source is down, or whether the interface itself is having problems. To resolve this uncertainty, the backup interface uses the path to the Historian Server to determine the status of the primary interface. If the primary interface is still communicating with the Historian Server, than failover to the backup is not required. However, if the primary interface is not posting data to the Historian Server, then the backup must initiate failover operations.

The primary interface also monitors the connection with the shared file to maintain the integrity of the failover configuration. If the primary interface can read and write to the shared file with no errors but the backup control information is not changing, then the backup is experiencing some error condition. To determine exactly where the problem exists, the primary interface uses the path to Historian to establish the status of the backup interface. For example, if the backup interface controls indicate that it has been shutdown, it may have been restarted and is now experiencing errors reading and writing to the shared file. Both primary and backup interfaces must always check their status through Historian to determine if one or the other is not updating the shared file and why.

Steady State Operation

Steady state operation is considered the normal operating condition. In this state, the primary interface is actively collecting data and sending its data to Historian. The primary interface is also updating its heartbeat value; monitoring the heartbeat value for the backup interface, checking the active ID value, and checking the device status for the backup interface every failover update interval on the shared file. Likewise, the backup interface is updating its heartbeat value; monitoring the heartbeat value for the primary interface, checking the active ID value, and checking the device status for the primary interface every failover update interval on the shared file. As long as the heartbeat value for the primary interface indicates that it is operating properly, the **ActiveID** has not changed, and the device status on the primary interface is good, the backup interface will continue in this mode of operation.

An interface configured for hot failover will have the backup interface actively collecting and queuing data but not sending that data to Historian. An interface for warm failover in the backup role is not actively collecting data from the data source even though it may be configured with Historian tags and may even have a good connection to the data source. An interface configured for cold failover in the backup role is not connected to the data source and upon initial startup will not have configured Historian tags.

The interaction between the interface and the shared file is fundamental to failover. The discussion that follows only refers to the data written to the shared file. However, every value written to the shared file is echoed to the tags on the Historian Server. Updating of the tags on the Historian Server is assumed to take place unless communication with the Historian Server is interrupted. The updates to the Historian Server will be buffered by Bufserv or PIBufss in this case.

In a hot failover configuration, each interface participating in the failover solution will queue three failover intervals worth of data to prevent any data loss. When a failover occurs, there may be a period of overlapping data for up to 3 intervals. The exact amount of overlap is determined by the timing and the cause of the failover and may be different every time. Using the default update interval of 5 seconds will result in overlapping data between 0 and 15 seconds. The no data loss claim for hot failover is based on a single point of failure. If both interfaces have trouble collecting data for the same period of time, data will be lost during that time.

As mentioned above, each interface has its own heartbeat value. In normal operation, the Heartbeat value on the shared file is incremented by UniInt from 1 - 15 and then wraps around to a value of 1 again. UniInt increments the heartbeat value on the shared file every failover update interval. The default failover update interval is 5 seconds. UniInt also reads the heartbeat value for the other interface copy participating in failover every failover update interval. If the connection to the Historian Server is lost, the value of the heartbeat will be incremented from 17 - 31 and then wrap around to a value of 17 again. Once the connection to the Historian Server is restored, the heartbeat values will revert back to the 1 - 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

During steady state, the **ActiveID** will equal the value of the failover ID of the primary interface. This value is set by UniInt when the interface enters the primary state and is not updated again by the primary interface until it shuts down gracefully. During shutdown, the primary interface will set the **ActiveID** to zero before shutting down. The backup interface has the ability to assume control as primary even if the current primary is not experiencing problems. This can be accomplished by setting the **ActiveID** tag on the Historian Server to the **ActiveID** of the desired interface copy.

As previously mentioned, in a hot failover configuration the backup interface actively collects data but does not send its data to Historian. To eliminate any data loss during a failover, the backup interface queues data in memory for three failover update intervals. The data in the queue is continuously updated to contain the most recent data. Data older than three update intervals is discarded if the primary interface is in a good status as determined by the backup. If the backup interface transitions to the primary, it will have data in its queue to send to Historian. This queued data is sent to Historian using the same function calls that would have been used had the interface been in a primary state when the function call was received from UniInt. If UniInt receives data without a timestamp, the primary copy uses the current Historian Time to timestamp data sent to Historian. Likewise, the backup copy timestamps data it receives without a timestamp with the current Historian Time before queuing its data. This preserves the accuracy of the timestamps.

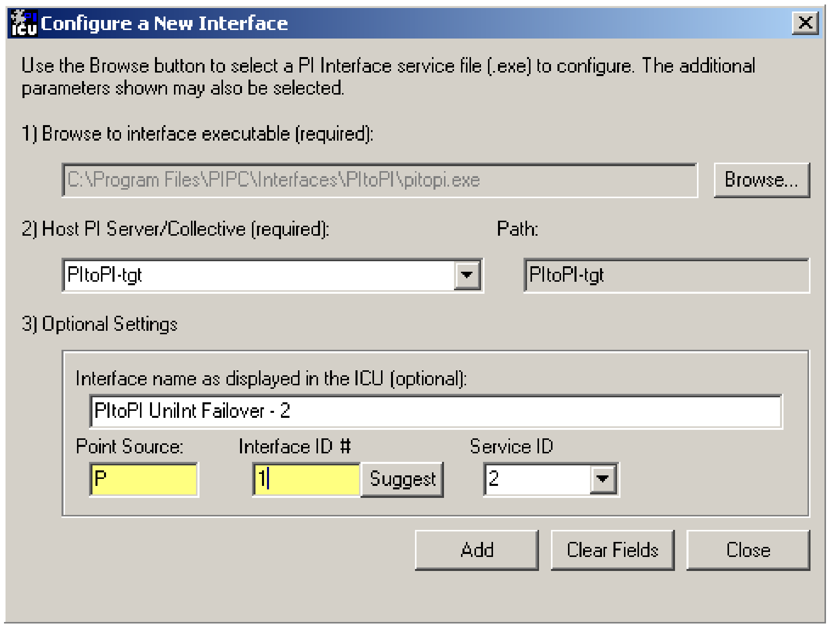
Failover Configuration Using ICU

The use of the ICU is the recommended and safest method for configuring the interface for UniInt failover. With the exception of the notes described in this section, the interface shall be configured with the ICU as described in the Configuring the interface with ICU section of this manual.

Note: With the exception of the **`/UFO_ID`** and **`/UFO_OtherID`** startup command-line parameters, the UniInt failover scheme requires that both copies of the interface have identical startup command files. This requirement causes the ICU to produce a message when creating the second copy of the interface stating that the "PS/ID combo already in use by the interface" as shown in Figure 5 below. Ignore this message and click the *Add* button.

Create the interface Instance with ICU

If the interface does not already exist in the ICU it must first be created. The procedure for doing this is the same as for non-failover interfaces. When configuring the second instance for UniInt Failover the Point Source and Interface ID will be in yellow and a message will be displayed saying this is already in use. This should be ignored.


Configure a New Interface

Use the Browse button to select a PI Interface service file (.exe) to configure. The additional parameters shown may also be selected.

1) Browse to interface executable (required):

C:\Program Files\PIPC\Interfaces\PItoPI\pitopi.exe

2) Host PI Server/Collective (required): PItoPI-tgt

Path: PItoPI-tgt

3) Optional Settings

Interface name as displayed in the ICU (optional): PItoPI UniInt Failover - 2

Point Source: P

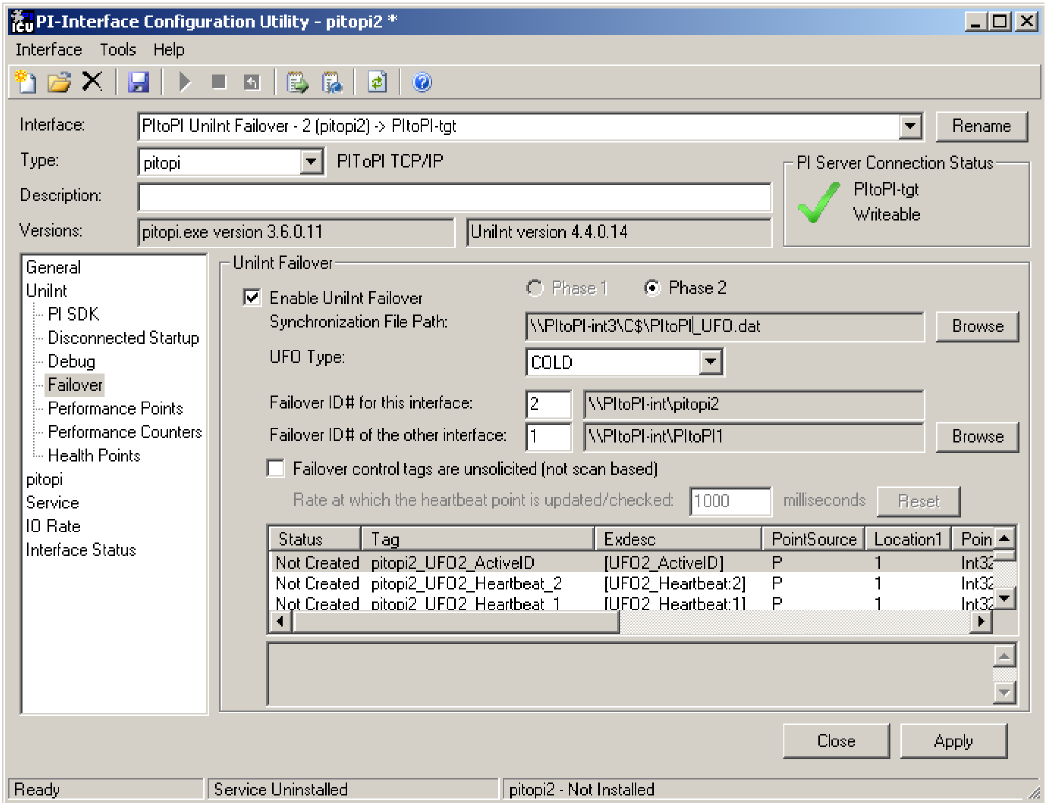
Interface ID #: 1

Service ID: 2

Figure 5: ICU configuration screen shows that the "PS/ID combo is already in use by the interface." The user must ignore the yellow boxes, which indicate errors, and click the *Add* button to configure the interface for failover.

Configuring the UniInt Failover Startup Parameters with ICU

There are three interface startup parameters that control UniInt failover: **`/UFO_ID`**, **`/UFO_OtherID`**, and **`/UFO_Interval`**. The `UFO` stands for UniInt Failover. The **`/UFO_ID`** and **`/UFO_OtherID`** parameters are required for the interface to operate in a failover configuration, but the **`/UFO_Interval`** is optional. Each of these parameters is described in detail in Configuring UniInt Failover through a Shared File (Phase 2) section and Start-Up Parameters.



Interface: PItoPI UniInt Failover - 2 (pitopi2) -> PItoPI-tgt

Type: pitopi   PIToPI TCP/IP

Description:

Versions: pitopi.exe version 3.6.0.11   UniInt version 4.4.0.14

PI Server Connection Status: PItoPI-tgt Writeable

General
UniInt
- PI SDK
- Disconnected Startup
- Debug
- Failover
- Performance Points
- Performance Counters
- Health Points
pitopi
Service
IO Rate
Interface Status

UniInt Failover

[✓] Enable UniInt Failover   Phase 1 [✓ Phase 2]

Synchronization File Path: \\PItoPI-int3\C$\PItoPI_UFO.dat

UFO Type: COLD

Failover ID# for this interface: 2   \\PItoPI-int\pitopi2

Failover ID# of the other interface: 1   \\PItoPI-int\PItoPI1

[ ] Failover control tags are unsolicited (not scan based)

Rate at which the heartbeat point is updated/checked: 1000 milliseconds

Status	Tag	Exdesc	PointSource	Location1	Poin
Not Created	pitopi2_UFO2_ActiveID	[UFO2_ActiveID]	P	1	Int32
Not Created	pitopi2_UFO2_Heartbeat_2	[UFO2_Heartbeat:2]	P	1	Int32
Not Created	pitopi2_UFO2_Heartbeat_1	[UFO2_Heartbeat:1]	P	1	Int32

Ready | Service Uninstalled | pitopi2 - Not Installed

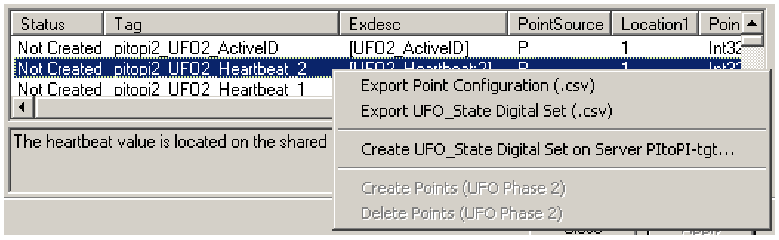
Figure 6: The figure above illustrates the ICU failover configuration screen showing the UniInt failover startup parameters (Phase 2). This copy of the interface defines its Failover ID as 2 (**`/UFO_ID=2`**) and the other interfaces Failover ID as 1 (**`/UFO_OtherID=1`**). The other failover interface copy must define its Failover ID as 1 (**`/UFO_ID=1`**) and the other interface Failover ID as 2 (**`/UFO_OtherID=2`**) in its ICU failover configuration screen. It also defines the location and name of the synchronization file as well as the type of failover as COLD.

Creating the Failover State Digital State Set

The UFO_State digital state set is used in conjunction with the failover state digital tag. If the UFO_State digital state set has not been created yet, it can be using either the Failover page of the ICU (1.4.1.0 or greater) or the Digital States plug-in in the SMT 3 Utility (3.0.0.7 or greater).

Using the ICU Utility to create Digital State Set

To use the *UniInt Failover* page to create the UFO_State digital state set right-click on any of the failover tags in the tag list and then select the *Create UFO_State Digital Set on Server XXXXXX…*, where XXXXXX is the Historian Server where the points will be or are create on.



Status	Tag	Exdesc	PointSource	Location1	Poin
Not Created	pitopi2_UFO2_ActiveID	[UFO2_ActiveID]	P	1	Int32
Not Created	pitopi2_UFO2_Heartbeat_2	[UFO2_Heartbeat:2]	P	1	Int32
Not Created	pitopi2_UFO2_Heartbeat_1				

The heartbeat value is located on the shared

Export Point Configuration (.csv)
Export UFO_State Digital Set (.csv)
Create UFO_State Digital Set on Server PItoPI-tgt…
Create Points (UFO Phase 2)
Delete Points (UFO Phase 2)

This choice will be grayed out if the UFO_State digital state set is already created on the XXXXXX Historian Server.

Using the SMT 3 Utility to create Digital State Set

Optionally, *Export UFO_State Digital Set (.csv)* can be selected to create a comma separated file to be imported via the System Management Tools (SMT3) (version 3.0.0.7 or higher) or use the `UniInt_Failover_DigitalSet_UFO_State.csv` file included in the installation kit.

The procedure below outlines the steps necessary to create a digital set on a Historian Sever using the “Import from File” function found in the SMT3 application. The procedure assumes the user has a basic understanding of the SMT3 application.

1. Open the SMT3 application.
2. Select the appropriate Historian Server from the Historian Servers window. If the desired server is not listed, add it using the Connection Manager. A view of the SMT application is shown in Figure 7 below.
3. From the *System Management Plug-Ins* window, select *Points* then *Digital States*. A list of available digital state sets will be displayed in the main window for the selected Historian Server. Refer to Figure 7 below.
4. In the main window, right-click on the desired server and select the *Import from File* option. Refer to Figure 7 below.

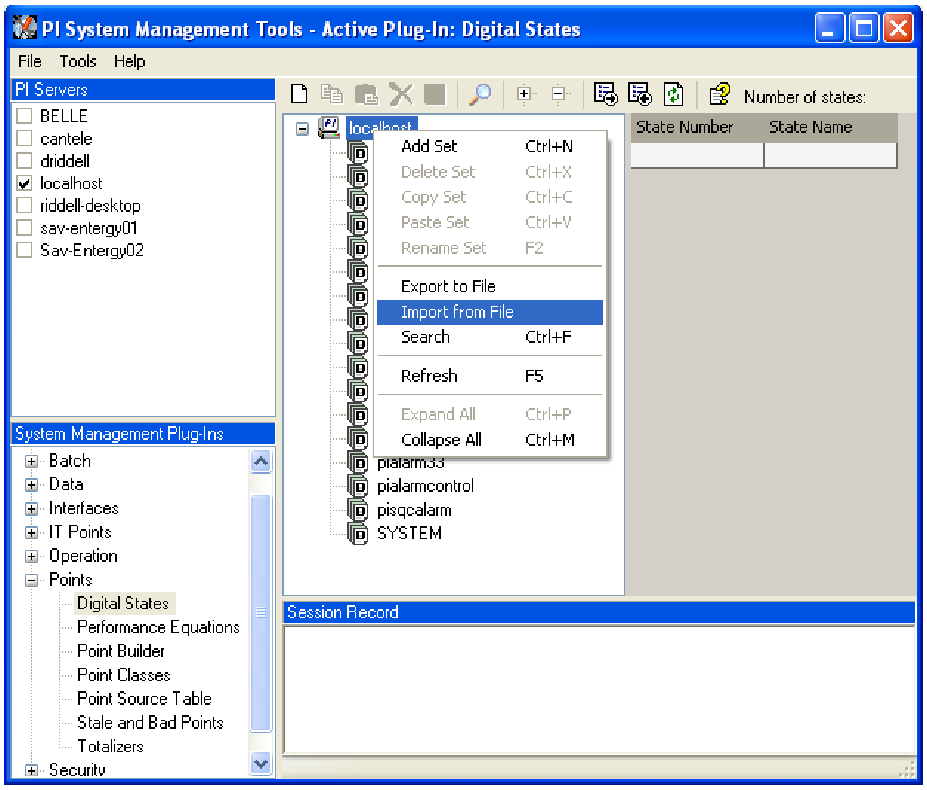
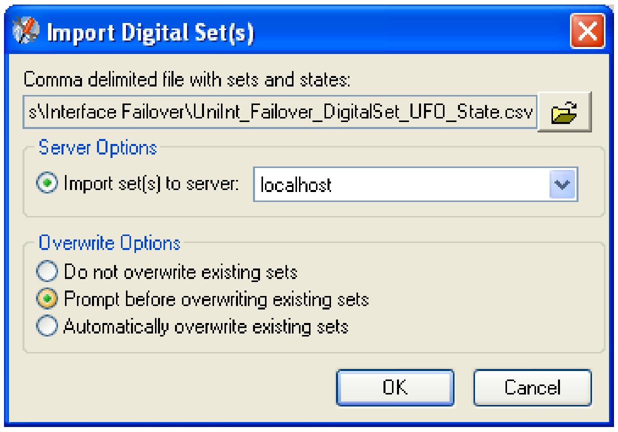


Figure 7: SMT application configured to import a digital state set file. The Historian Servers window shows the “localhost” Historian Server selected along with the System Management Plug-Ins window showing the *Digital States* Plug-In as being selected. The digital state set file can now be imported by selecting the *Import from File* option for the localhost.

5. Navigate to and select the `UniInt_Failover_DigitalSet_UFO_State.csv` file for import using the Browse icon on the display. Select the desired *Overwrite Options*. Click on the *OK* button. Refer to Figure 8 below.


Import Digital Set(s)

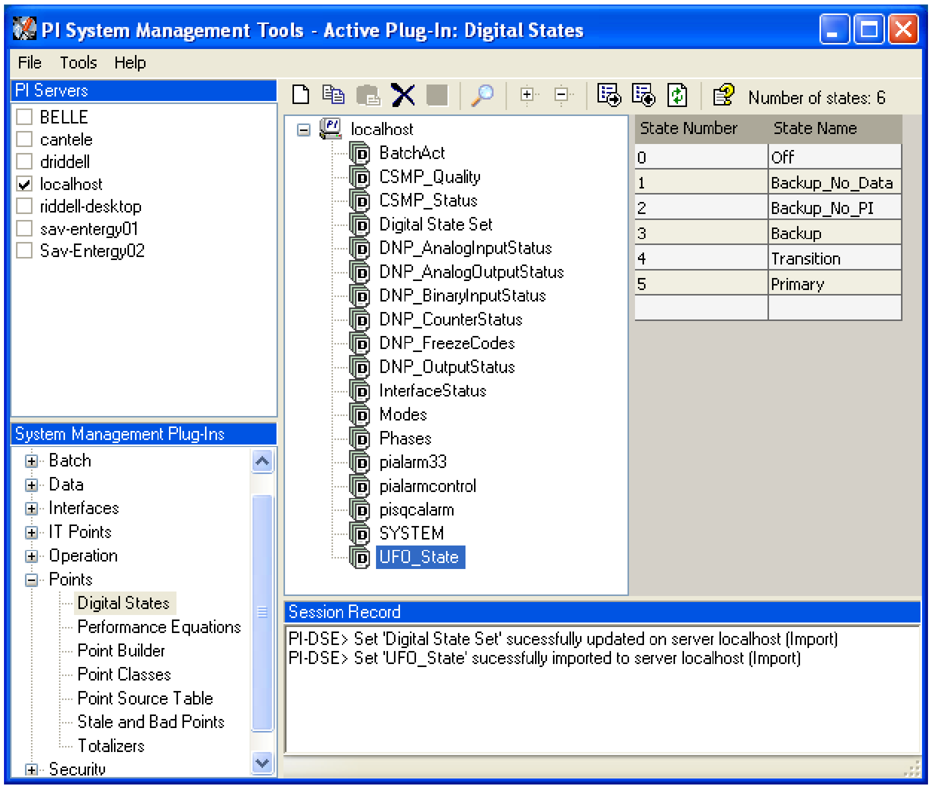
Comma delimited file with sets and states:
s\Interface Failover\UniInt_Failover_DigitalSet_UFO_State.csv

Server Options
[✓ Import set(s) to server:] localhost

Overwrite Options
Do not overwrite existing sets
[✓ Prompt before overwriting existing sets]
Automatically overwrite existing sets

Figure 8: SMT application *Import Digital Set(s)* window. This view shows the `UniInt_Failover_DigitalSet_UFO_State.csv` file as being selected for import. Select the desired *Overwrite Options* by choosing the appropriate radio button.

6. The `UFO_State` digital set is created as shown in Figure 9 below.



State Number	State Name
0	Off
1	Backup_No_Data
2	Backup_No_PI
3	Backup
4	Transition
5	Primary

```
PI-DSE> Set 'Digital State Set' sucessfully updated on server localhost (Import)
PI-DSE> Set 'UFO_State' sucessfully imported to server localhost (Import)
```

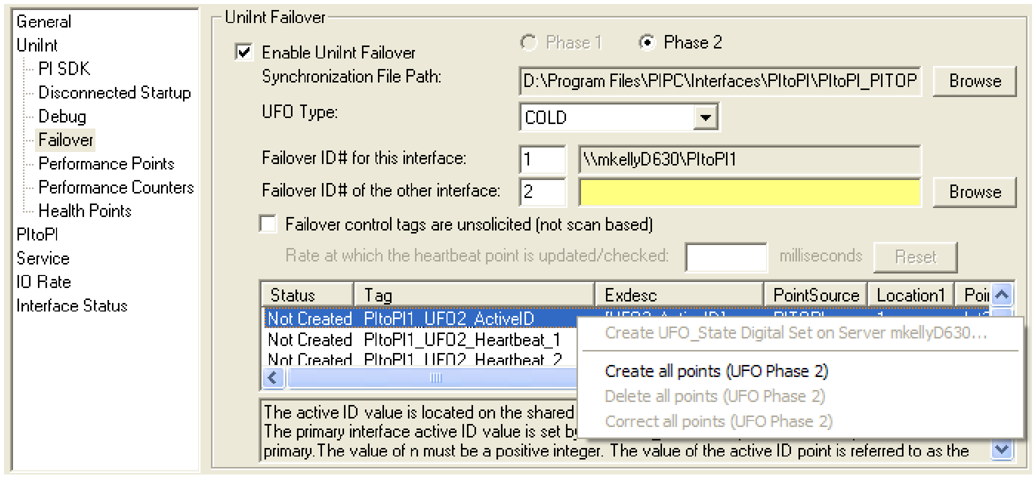
Figure 9: The SMT application showing the `UFO_State` digital set created on the "localhost" Historian Server.

Creating the UniInt Failover Control and Failover State Tags (Phase 2)

The ICU can be used to create the UniInt Failover Control and State Tags.

To use the ICU *Failover* page to create these tags simply right-click any of the failover tags in the tag list and select the *Create all points (UFO Phase 2)* menu item.

If this menu choice is grayed out it is because the UFO_State digital state set has not been created on the Server yet. There is a menu choice *Create UFO_State Digitial Set on Server xxxxxxx…* which can be used to create that digital state set. Once this has been done then the *Create all points (UFO Phase2)* command should be available.



UniInt Failover

Phase 1 [✓ Phase 2]

[✓] Enable UniInt Failover

Synchronization File Path: D:\Program Files\PIPC\Interfaces\PItoPI\PItoPI_PITOP

UFO Type: COLD

Failover ID# for this interface: 1 \\mkellyD630\PItoPI1

Failover ID# of the other interface: 2

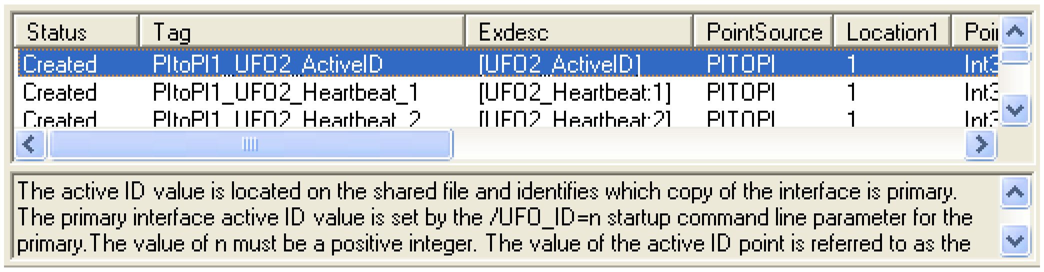
[ ] Failover control tags are unsolicited (not scan based)

Rate at which the heartbeat point is updated/checked: milliseconds

Status	Tag	Exdesc	PointSource	Location1	Poi
Not Created	PItoPI1_UFO2_ActiveID				
Not Created	PItoPI1_UFO2_Heartbeat_1				
Not Created	PItoPI1_UFO2_Heartbeat_2				

The active ID value is located on the shared
The primary interface active ID value is set b
primary. The value of n must be a positive integer. The value of the active ID point is referred to as the

Once the failover control and failover state tags have been created the *Failover* page of the ICU should look similar to the illustration below.

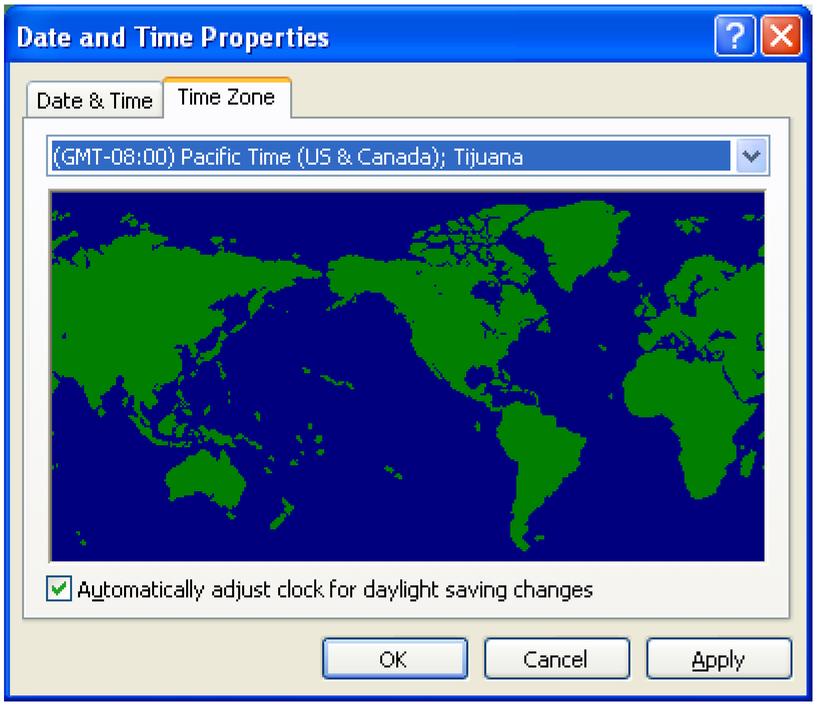


Status	Tag	Exdesc	PointSource	Location1	Poi
Created	PItoPI1_UFO2_ActiveID	[UFO2_ActiveID]	PITOPI	1	Int3
Created	PItoPI1_UFO2_Heartbeat_1	[UFO2_Heartbeat:1]	PITOPI	1	Int3
Created	PItoPI1_UFO2_Heartbeat_2	[UFO2_Heartbeat:2]	PITOPI	1	Int3

The active ID value is located on the shared file and identifies which copy of the interface is primary.
The primary interface active ID value is set by the /UFO_ID=n startup command line parameter for the primary. The value of n must be a positive integer. The value of the active ID point is referred to as the

Chapter 10. 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 *Automatically adjust clock for daylight saving changes* box. For example,



(GMT-08:00) Pacific Time (US & Canada); Tijuana

[✓] Automatically adjust clock for daylight saving changes

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 Variables* button under the *Advanced* tab, and remove TZ from the list of environment variables.

Chapter 11. Security

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

Note that the Trust Database, which is maintained by the Base Subsystem, replaces the Proxy Database used prior to Historian 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 *Historian Server System Management Guide*.

If the interface cannot write data to the Historian 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 Server, it writes a `-999` error. See the section Appendix A: Error and Informational Messages for additional information on error messaging.

Historian Server v3.3 and Higher

Security configuration using piconfig

For Historian Server v3.3 and higher, the following example demonstrates how to edit the 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 System Management Tools 3.x may also be used to edit the Trust table.

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

Historian Server v3.2

For Historian Server v3.2, the following example demonstrates how to edit the Historian 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 interface node *as it is seen by Historian Server*.

Tag and Node Security

The receiving Historian Server must allow the interface tag attribute read access and data read and write access. The source Historian Server must allow the interface tag attribute read access and data read access for source tags.

System Manager Responsibilities

Securing the source archive data requires that both the source and receiving system managers perform the following tasks.

Receiving System Manager

1. Give the source system manager a copy of the FactoryTalk Historian to Historian Interface manual.
2. Edit the PItoPI.ini file and add the keys SecurityFile, SourceLogin, SourcePassword under the section [PITOPI-#] where “#” is the interface identification number. The three security keys may be listed under a particular scan class if the scan is retrieving data from a PI 2 server different from the default source server (use section name [PITOPI-#.*] where “*” is the scan class number).

   Alternately, edit the startup script to contain the parameters /ln=pidemo /pw=pidemo replacing pidemo with the PI user name and password under which the data will be accessed. Set the name of the security file: for example /sf=test.

Source System Manager

1. The file *PISysDat:PIServer.dat* will be used to provide overall security for the FactoryTalk Historian database. In this file, the receiving node must be given at least read-login access.
2. Create and maintain a security file for each FactoryTalk Historian to Historian interface that will read data from the source system. This file is *PISYSDAT:PITOPITEST.SEC*, where `TEST` is the file name specified on the receiving node PIToPI command line.
3. A PI username and password must be provided which the interface will use as its Historian login.

Note: Revisions to the security file are checked every 30 minutes.

Sample Security Files

Sample 1

```
I!  Mode is set to include.
T!  Specification type set to tag masks.
!   Default to including all points in system.*
X!  Reset mode to exclude.
T!
!   All tags ending in '.al' are excluded.
*.al
unit?flow.*
S!  Point source specifications.
?   ! All fractal tags are excluded.
E!
```

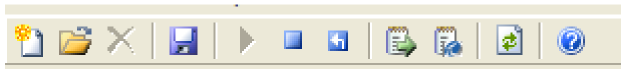
Sample 2

```
X!  Mode is set to exclude.
T!  Tag specifications.
!   Default to excluding all tags not explicitly matched.*
I!
T!
!   Some tag masks with wildcards.
!   multiple wildcards may be used.
cd*
*unit1*
!   Match two of any characters.
reactor1.??
!   Some tag names.
Sinusoid
le420
S!  Point source specifications.
!   All PE tags included.
C
!   Only point source 'P' with Location1 = 2 included.
P2
```

```
E!
```

Chapter 12. Starting / Stopping the interface

This section describes starting and stopping the interface once it has been installed as a service. See the *UniInt Interface User Manual* to run the interface interactively.



Starting Interface as a Service

If the interface was installed as service, it can be started from ICU, the Services control panel or with the command:

```
PItoPI.exe -start
```

To start the interface service with ICU, use the ▶ button on the ICU toolbar.

A message will inform the user 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` file. Verify that the root name of the `.bat` file and the `.exe` file are the same, and that the `.bat` 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 Interface Running as a Service

If the interface was installed as service, it can be stopped at any time from ICU, the Services control panel or with the command:

```
PItoPI.exe -stop
```

The service can be removed by:

```
PItoPI.exe -remove
```

To stop the interface service with ICU, use the ■ button on the ICU toolbar.

Chapter 13. Buffering

Buffering refers to an interface node's ability to temporarily store the data that interfaces collect and to forward these data to the appropriate Historian Servers. Rockwell Automation strongly recommends that you enable buffering on your interface nodes. Otherwise, if the interface node stops communicating with the Historian Server, you lose the data that your interfaces collect.

The PI SDK installation kit installs two buffering applications: the PI Buffer Subsystem (PIBufss) and the PI API Buffer Server (Bufserv). PIBufss and Bufserv are mutually exclusive; that is, on a particular computer, you can run only one of them at any given time.

If you have Historian Servers that are part of a Historian Collective, PIBufss supports *n-way buffering*. N-way buffering refers to the ability of a buffering application to send the same data to each of the Historian Servers in a Historian Collective. (Bufserv also supports n-way buffering, but Rockwell Automation recommends that you run PIBufss instead.)

Which Buffering Application to Use

You should use PIBufss whenever possible because it offers better throughput than Bufserv. In addition, if the interfaces on an interface node are sending data to a Historian Collective, PIBufss guarantees identical data in the archive records of all the Historian Servers that are part of that collective.

You can use PIBufss only under the following conditions:

- the Historian Server version is at least 2.x.x; and
- all of the interfaces running on the interface node send data to the same Historian Server or to the same Historian Collective.

If any of the following scenarios apply, you must use Bufserv:

- the Historian Server version is earlier than 2.x.x; or
- the interface node runs multiple interfaces, and these interfaces send data to multiple Historian Servers that are not part of a single Historian Collective.

If an interface node runs multiple interfaces, and these interfaces send data to two or more Historian Collectives, then neither PIBufss nor Bufserv is appropriate. The reason is that PIBufss and Bufserv can buffer data only to a single collective. If you need to buffer to more than one Historian Collective, you need to use two or more interface nodes to run your interfaces.

It is technically possible to run Bufserv on the Historian Server Node. However, Rockwell Automation does not recommend this configuration.

How Buffering Works

A complete technical description of PIBufss and Bufserv is beyond the scope of this document. However, the following paragraphs provide some insights on how buffering works.

When an interface node has buffering enabled, the buffering application (PIBufss or Bufserv) connects to the Historian Server. It also creates shared memory storage.

When an interface program makes a PI API function call that writes data to the Historian Server (for example, `pisn_sendexceptionqx()`), the PI API checks whether buffering is enabled. If it is, these data writing functions do not send the interface data to the Historian Server. Instead, they write the data to the shared memory storage that the buffering application created.

The buffering application (either Bufserv or PIBufss) in turn

- reads the data in shared memory, and
- if a connection to the Historian Server exists, sends the data to the Historian Server; or
- if there is no connection to the Historian Server, continues to store the data in shared memory (if shared memory storage is available) or writes the data to disk (if shared memory storage is full).

When the buffering application re-establishes connection to the Historian Server, it writes to the Historian Server the interface data contained in both shared memory storage and disk.

(Before sending data to the Historian Server, PIBufss performs further tasks such data validation and data compression, but the description of these tasks is beyond the scope of this document.)

When PIBufss writes interface data to disk, it writes to multiple files. The names of these buffering files are `PIBUFQ_*.DAT`.

When Bufserv writes interface data to disk, it writes to a single file. The name of its buffering file is `APIBUF.DAT`.

As a previous paragraph indicates, PIBufss and Bufserv create shared memory storage at startup. These memory buffers must be large enough to accommodate the data that an interface collects during a single scan. Otherwise, the interface may fail to write all its collected data to the memory buffers, resulting in data loss. The buffering configuration section of this chapter provides guidelines for sizing these memory buffers.

When buffering is enabled, it affects the entire interface node. That is, you do not have a scenario whereby the buffering application buffers data for one interface running on an interface node but not for another interface running on the same interface node.

Buffering and Historian Server Security

After you enable buffering, it is the buffering application—and not the interface program—that writes data to the Historian Server. If the Historian Server’s trust table contains a trust entry that allows all applications on an interface node to write data, then the buffering application is able write data to the Historian Server.

However, if the Historian Server contains an interface-specific Trust entry that allows a particular interface program to write data, you must have a Trust entry specific to buffering. The following are the appropriate entries for the Application Name field of a Trust entry:

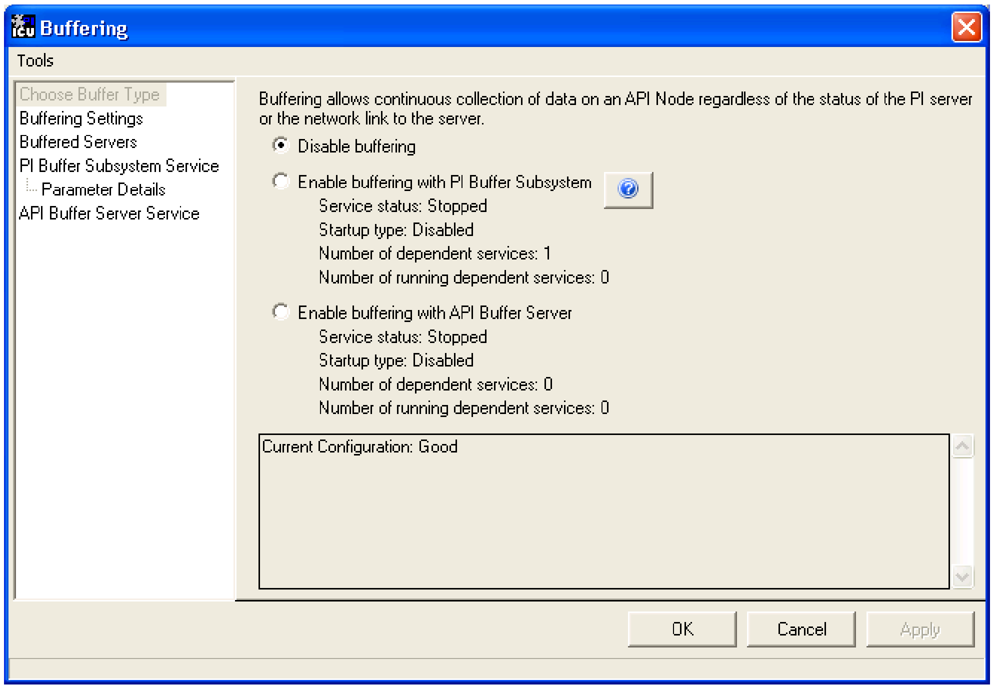
Buffering Application	Application Name field for Trust
Buffer Subsystem	PIBufss.exe
PI API Buffer Server	APIBE (if the PI API is using 4 character process names) APIBUF (if the PI API is using 8 character process names)

To use a process name greater than 4 characters in length for a trust application name, use the LONGAPPNAME=1 in the PIClient.ini file.

Enabling Buffering on an Interface Node with the ICU

The ICU allows you to select either PIBufss or Bufserv as the buffering application for your interface node. Run the ICU and select *Tools* > *Buffering*.

Choose Buffer Type



Buffering allows continuous collection of data on an API Node regardless of the status of the PI server or the network link to the server.

[✓ Disable buffering]

Enable buffering with PI Buffer Subsystem
Service status: Stopped
Startup type: Disabled
Number of dependent services: 1
Number of running dependent services: 0

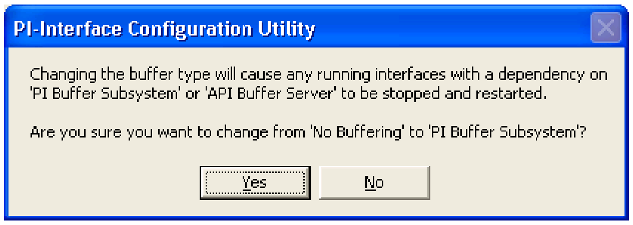
Enable buffering with API Buffer Server
Service status: Stopped
Startup type: Disabled
Number of dependent services: 0
Number of running dependent services: 0

Current Configuration: Good

To select PIBufss as the buffering application, choose *Enable buffering with PI Buffer Subsystem*.

To select Bufserv as the buffering application, choose *Enable buffering with API Buffer Server*.

If a warning message such as the following appears, click *Yes*.



PI-Interface Configuration Utility

Changing the buffer type will cause any running interfaces with a dependency on 'PI Buffer Subsystem' or 'API Buffer Server' to be stopped and restarted.

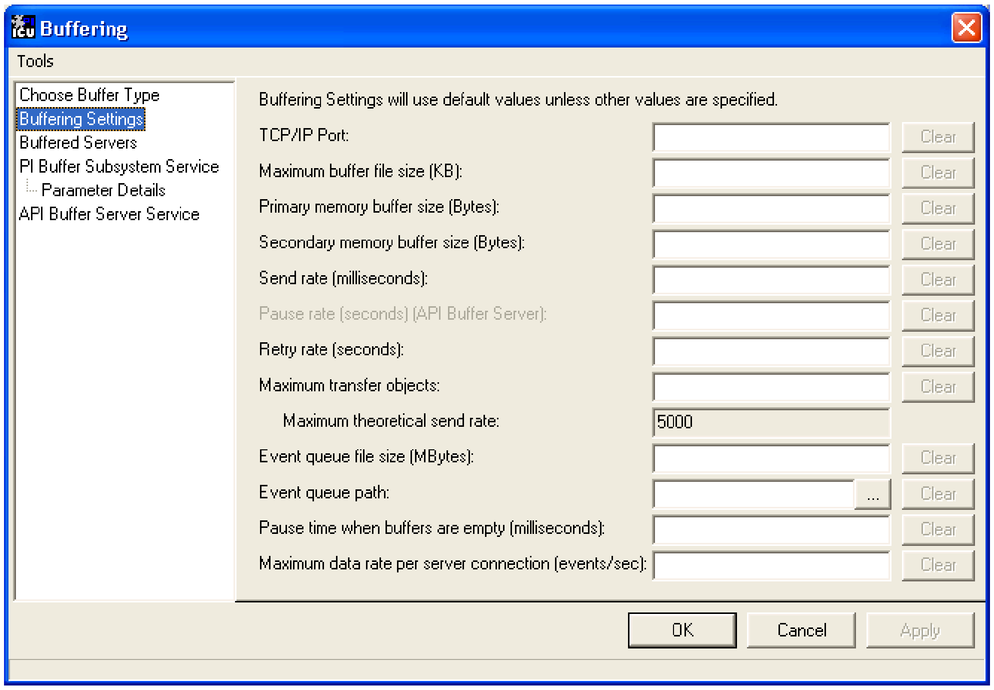
Are you sure you want to change from 'No Buffering' to 'PI Buffer Subsystem'?

Buffering Settings

There are a number of settings that affect the operation of PIBufss and Bufserv. The *Buffering Settings* page allows you to set these parameters. If you do not enter values for these parameters, PIBufss and Bufserv use default values.

PIBufss

For PIBufss, the paragraphs below describe the settings that may require user intervention. Please contact Rockwell Automation Technical Support for assistance in further optimizing these and all remaining settings.



Buffering Settings will use default values unless other values are specified.

TCP/IP Port:	
Maximum buffer file size (KB):	
Primary memory buffer size (Bytes):	
Secondary memory buffer size (Bytes):	
Send rate (milliseconds):	
Pause rate (seconds) (API Buffer Server):	
Retry rate (seconds):	
Maximum transfer objects:	
Maximum theoretical send rate:	5000
Event queue file size (MBytes):	
Event queue path:	
Pause time when buffers are empty (milliseconds):	
Maximum data rate per server connection (events/sec):	

Primary and Secondary Memory Buffer Size (Bytes)

This is a key parameter for buffering performance. The sum of these two memory buffer sizes must be large enough to accommodate the data that an interface collects during a single scan. A typical event with a Float32 point type requires about 25 bytes. If an interface writes data to 5,000 points, it can potentially send 125,000 bytes (25 * 5000) of data in one scan. As a result, the size of each memory buffer should be 62,500 bytes.

The default value of these memory buffers is 32,768 bytes. Rockwell Automation recommends that these two memory buffer sizes should be increased to the maximum of 2000000 for the best buffering performance.

Send rate (milliseconds)

Send rate is the time in milliseconds that PIBufss waits between sending up to the *Maximum transfer objects* (described below) to the Historian Server. The default value is 100. The valid range is 0 to 2,000,000.

Maximum transfer objects

Maximum transfer objects is the maximum number of events that PIBufss sends between each *Send rate* pause. The default value is 500. The valid range is 1 to 2,000,000.

Event Queue File Size (Mbytes)

This is the size of the event queue files. PIBufss stores the buffered data to these files. The default value is 32. The range is 8 to 131072 (8 to 128 Gbytes). Please see the section entitled, "Queue File Sizing" in the *pibufss.chm* file for details on how to appropriately size the event queue files.

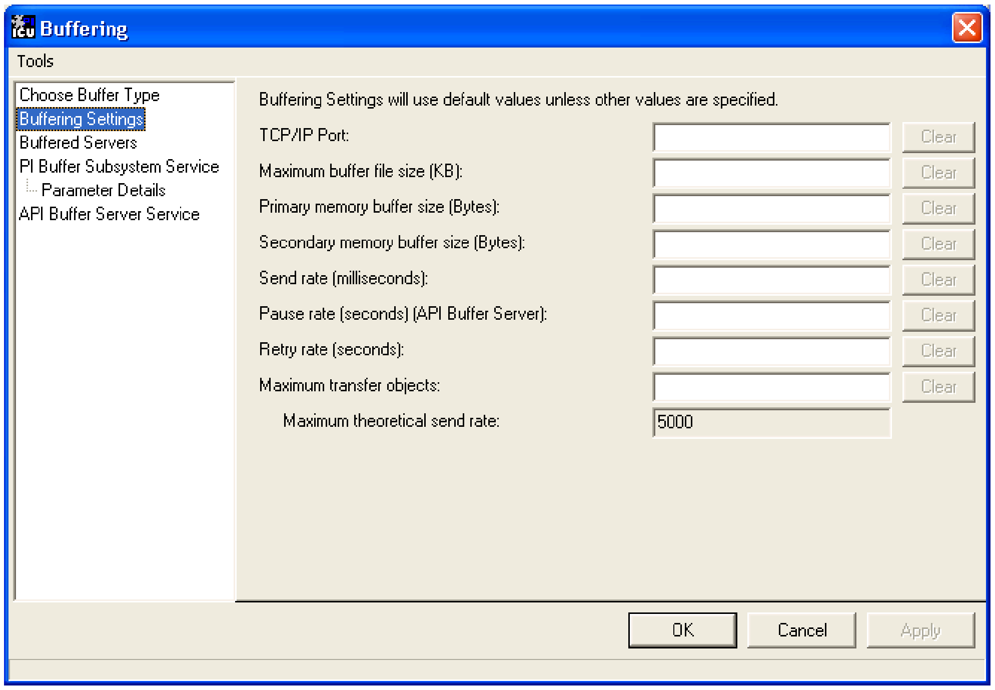
Event Queue Path

This is the location of the event queue file. The default value is `[PIHOME]\DAT`.

For optimal performance and reliability, Rockwell Automation recommends that you place the PIBufss event queue files on a different drive/controller from the system drive and the drive with the Windows paging file. (By default, these two drives are the same.)

Bufserv

For Bufserv, the paragraphs below describe the settings that may require user intervention. Please contact Rockwell Automation Technical Support for assistance in further optimizing these and all remaining settings.



Buffering Settings will use default values unless other values are specified.

TCP/IP Port:	
Maximum buffer file size (KB):	
Primary memory buffer size (Bytes):	
Secondary memory buffer size (Bytes):	
Send rate (milliseconds):	
Pause rate (seconds) (API Buffer Server):	
Retry rate (seconds):	
Maximum transfer objects:	
Maximum theoretical send rate:	5000

Maximum buffer file size (KB)

This is the maximum size of the buffer file (`[PIHOME]\DAT\APIBUF.DAT`). When Bufserv cannot communicate with the Historian Server, it writes and appends data to this file. When the buffer file reaches this maximum size, Bufserv discards data.

The default value is 2,000,000 KB, which is about 2 GB. The range is from 1 to 2,000,000.

Primary and Secondary Memory Buffer Size (Bytes)

This is a key parameter for buffering performance. The sum of these two memory buffer sizes must be large enough to accommodate the data that an interface collects during a single scan. A typical event with a Float32 point type requires about 25 bytes. If an interface writes data to 5,000 points, it can potentially send 125,000 bytes (25 * 5000) of data in one scan. As a result, the size of each memory buffer should be 62,500 bytes.

The default value of these memory buffers is 32,768 bytes. Rockwell Automation recommends that these two memory buffer sizes should be increased to the maximum of 2000000 for the best buffering performance.

Send rate (milliseconds)

Send rate is the time in milliseconds that Bufserv waits between sending up to the *Maximum transfer objects* (described below) to the Historian Server. The default value is 100. The valid range is 0 to 2,000,000.

Maximum transfer objects

Maximum transfer objects is the maximum number of events that Bufserv sends between each *Send rate* pause. The default value is 500. The valid range is 1 to 2,000,000.

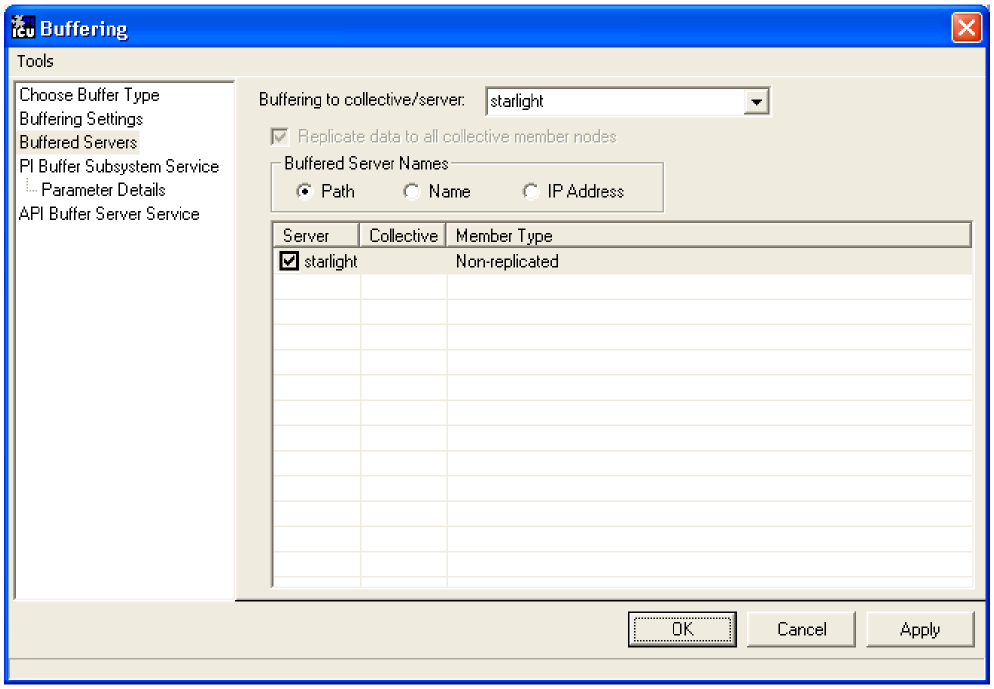
Buffered Servers

The *Buffered Servers* page allows you to define the Historian Servers or Historian Collective that the buffering application writes data.

PIBufss

PIBufss buffers data only to a single Historian Server or a Historian Collective. Select the Historian Server or the Historian Collective from the *Buffering to collective/server* drop down list box.

The following screen shows that PIBufss is configured to write data to a standalone Historian Server named `starlight`. Notice that the *Replicate data to all collective member nodes* check box is disabled because this Historian Server is not part of a collective. (PIBufss automatically detects whether a Historian Server is part of a collective.)



Buffering to collective/server: starlight

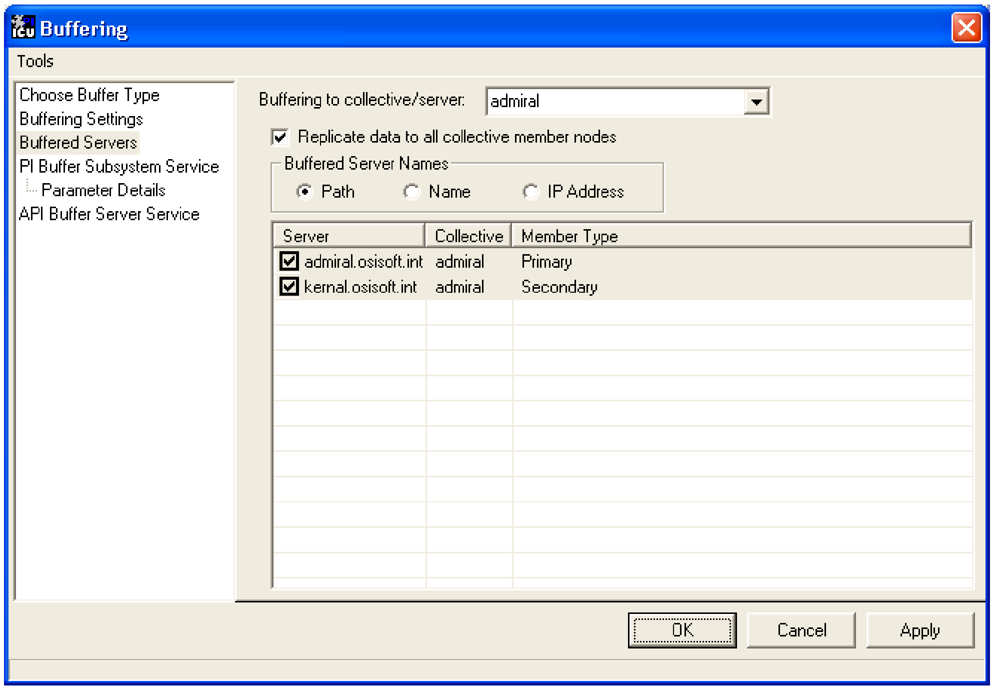
[✓] Replicate data to all collective member nodes

Buffered Server Names: [● Path] ○ Name ○ IP Address

Server	Collective	Member Type
☑ starlight		Non-replicated

The following screen shows that PIBufss is configured to write data to a Historian Collective named `admiral`. By default, PIBufss replicates data to all collective members. That is, it provides n-way buffering.

You can override this option by not checking the *Replicate data to all collective member nodes* check box. Then, uncheck (or check) the Historian Server collective members as desired.



Buffering to collective/server: admiral

[✓] Replicate data to all collective member nodes

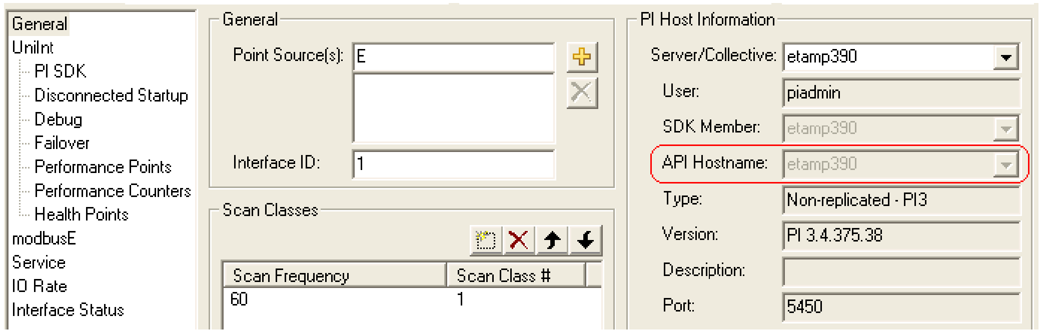
Buffered Server Names: [● Path] ○ Name ○ IP Address

Server	Collective	Member Type
☑ admiral.osisoft.int	admiral	Primary
☑ kernal.osisoft.int	admiral	Secondary

Bufserv

Bufserv buffers data to a standalone Historian Server, or to multiple standalone Historian Servers. (If you want to buffer to multiple Historian Servers that are part of a Historian Collective, you should use PIBufss.)

If the Historian Server to which you want Bufserv to buffer data is not in the Server list, enter its name in the *Add a server* box and click the *Add Server* button. This Historian Server name must be identical to the *API Hostname* entry:

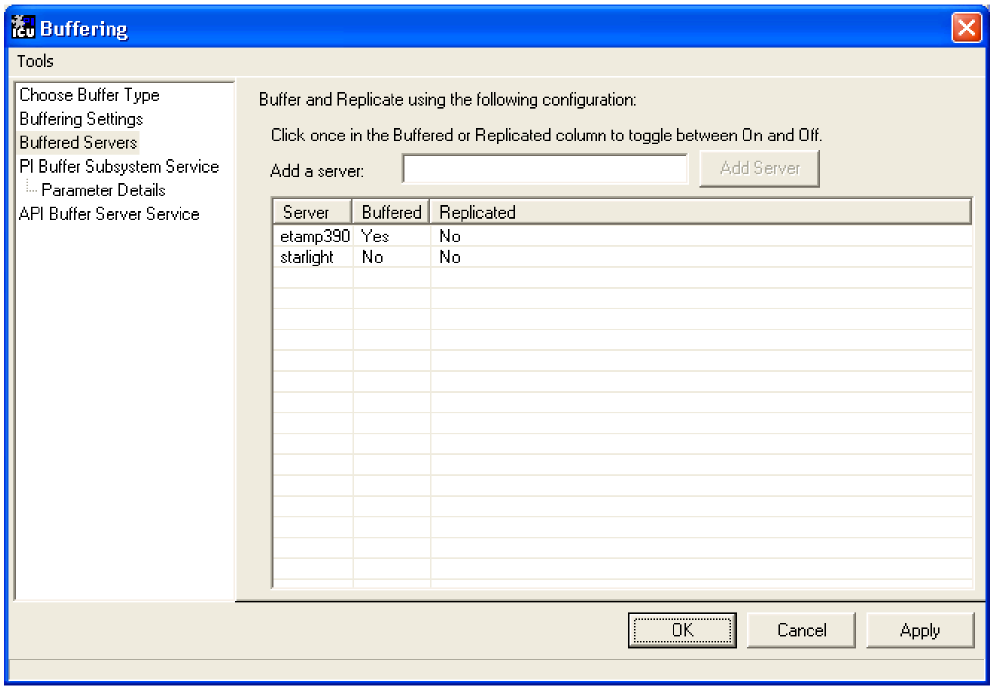


Point Source(s): E
Interface ID: 1

Scan Frequency	Scan Class #
60	1

Server/Collective: etamp390
User: piadmin
SDK Member: etamp390
API Hostname: etamp390
Type: Non-replicated - PI3
Version: PI 3.4.375.38
Description:
Port: 5450

The following screen shows that Bufserv is configured to write to a standalone Historian Server named `etamp390`. You use this configuration when all the interfaces on the interface node write data to `etamp390`.



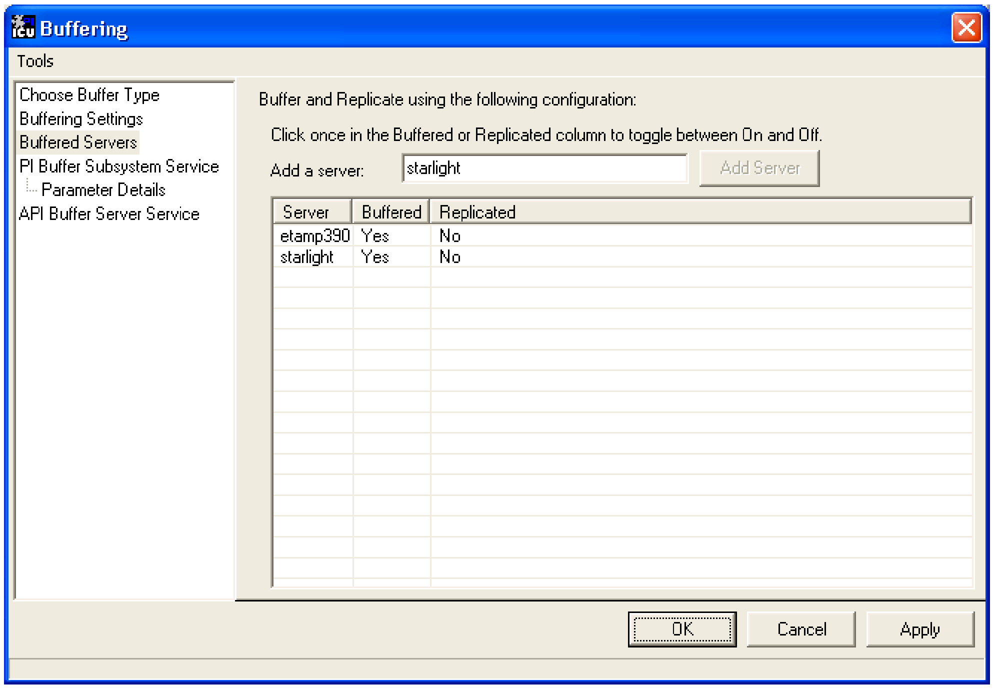
Buffer and Replicate using the following configuration:

Click once in the Buffered or Replicated column to toggle between On and Off.

Add a server:

Server	Buffered	Replicated
etamp390	Yes	No
starlight	No	No

The following screen shows that Bufserv is configured to write to two standalone Historian Servers, one named `etamp390` and the other one named `starlight`. You use this configuration when some of the interfaces on the interface node write data to `etamp390` and some write to `starlight`.



Buffer and Replicate using the following configuration:

Click once in the Buffered or Replicated column to toggle between On and Off.

Add a server: starlight

Server	Buffered	Replicated
etamp390	Yes	No
starlight	Yes	No

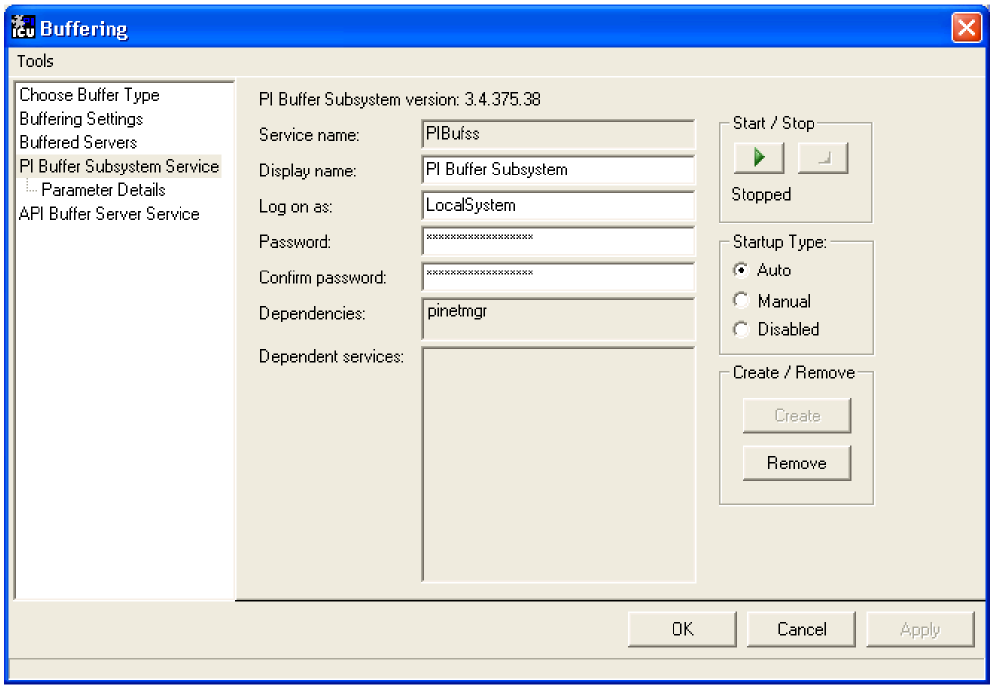
Installing Buffering as a Service

Both the PIBufss and Bufserv applications run as a Service.

Buffer Subsystem Service

Use the *Buffer Subsystem Service* page to configure PIBufss as a Service. This page also allows you to start and stop the PIBufss service.

PIBufss does not require the logon rights of the local administrator account. It is sufficient to use the LocalSystem account instead. Although the screen below shows asterisks for the LocalSystem password, this account does not have a password.



PI Buffer Subsystem version: 3.4.375.38

Service name: PIBufss

Display name: PI Buffer Subsystem

Log on as: LocalSystem

Password: ********************

Confirm password: ********************

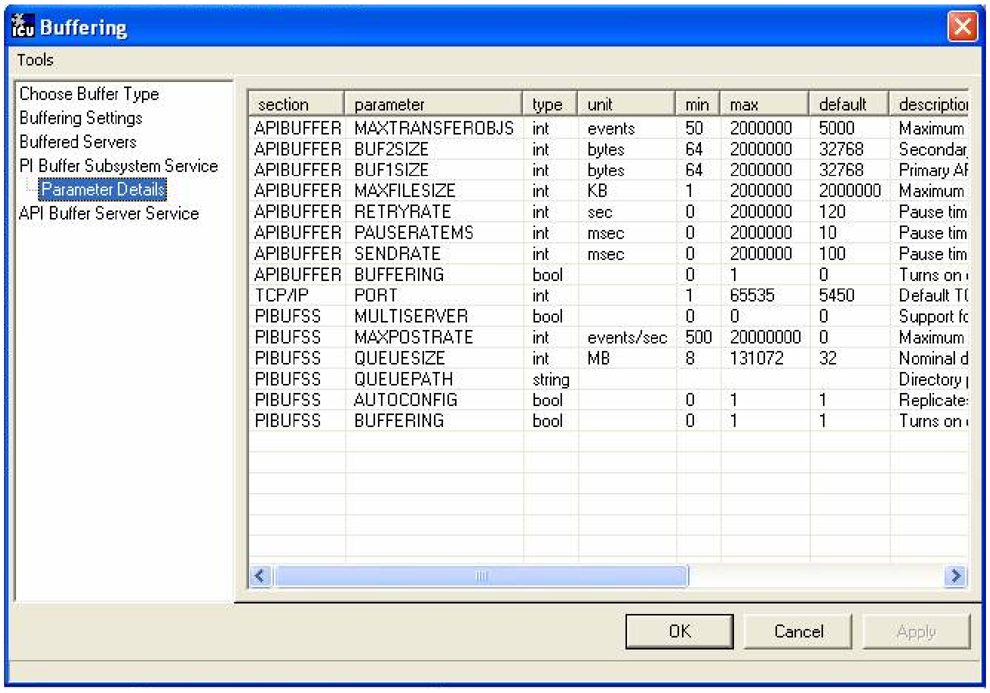
Dependencies: pinetmgr

Dependent services:

Start / Stop: Stopped

Startup Type: [✓ Auto] Manual Disabled

Create / Remove: Create Remove

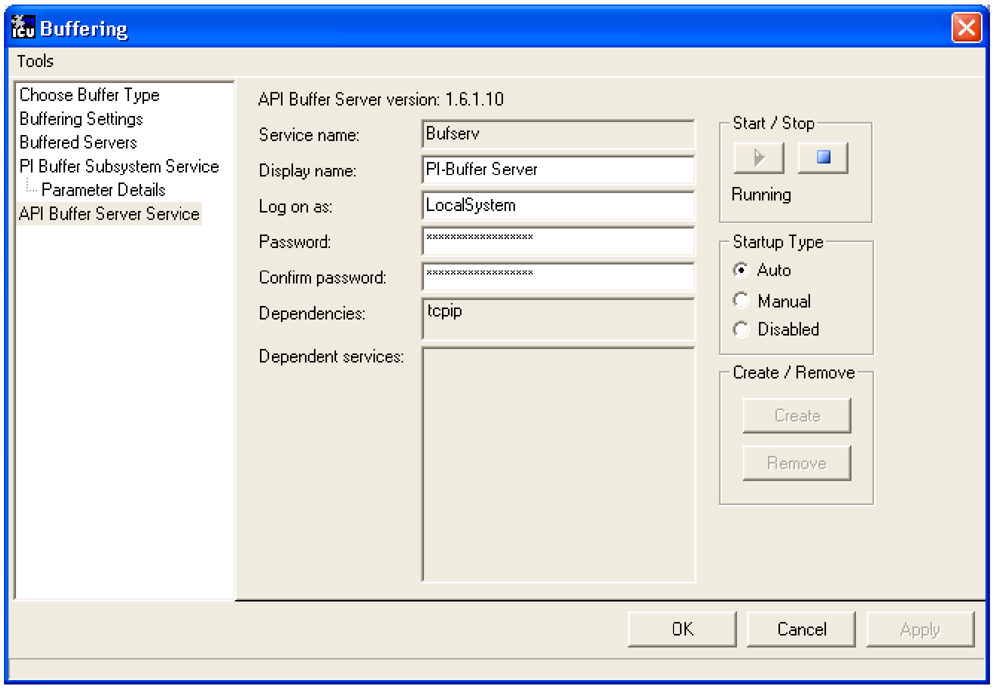


section	parameter	type	unit	min	max	default	descriptio
APIBUFFER	MAXTRANSFEROBJS	int	events	50	2000000	5000	Maximum
APIBUFFER	BUF2SIZE	int	bytes	64	2000000	32768	Secondar
APIBUFFER	BUF1SIZE	int	bytes	64	2000000	32768	Primary AF
APIBUFFER	MAXFILESIZE	int	KB	1	2000000	2000000	Maximum
APIBUFFER	RETRYRATE	int	sec	0	2000000	120	Pause tim
APIBUFFER	PAUSERATEMS	int	msec	0	2000000	10	Pause tim
APIBUFFER	SENDRATE	int	msec	0	2000000	100	Pause tim
APIBUFFER	BUFFERING	bool		0	1	0	Turns on
TCP/IP	PORT	int		1	65535	5450	Default T(
PIBUFSS	MULTISERVER	bool		0	0	0	Support fc
PIBUFSS	MAXPOSTRATE	int	events/sec	500	20000000	0	Maximum
PIBUFSS	QUEUESIZE	int	MB	8	131072	32	Nominal d
PIBUFSS	QUEUEPATH	string					Directory
PIBUFSS	AUTOCONFIG	bool		0	1	1	Replicate
PIBUFSS	BUFFERING	bool		0	1	1	Turns on

API Buffer Server Service

Use the *API Buffer Server Service* page to configure Bufserv as a Service. This page also allows you to start and stop the Bufserv Service

Bufserv version 1.6 and later does not require the logon rights of the local administrator account. It is sufficient to use the LocalSystem account instead. Although the screen below shows asterisks for the LocalSystem password, this account does not have a password.



API Buffer Server version: 1.6.1.10

Service name: Bufserv

Display name: PI-Buffer Server

Log on as: LocalSystem

Password: ******************

Confirm password: ******************

Dependencies: tcpip

Dependent services:

Running

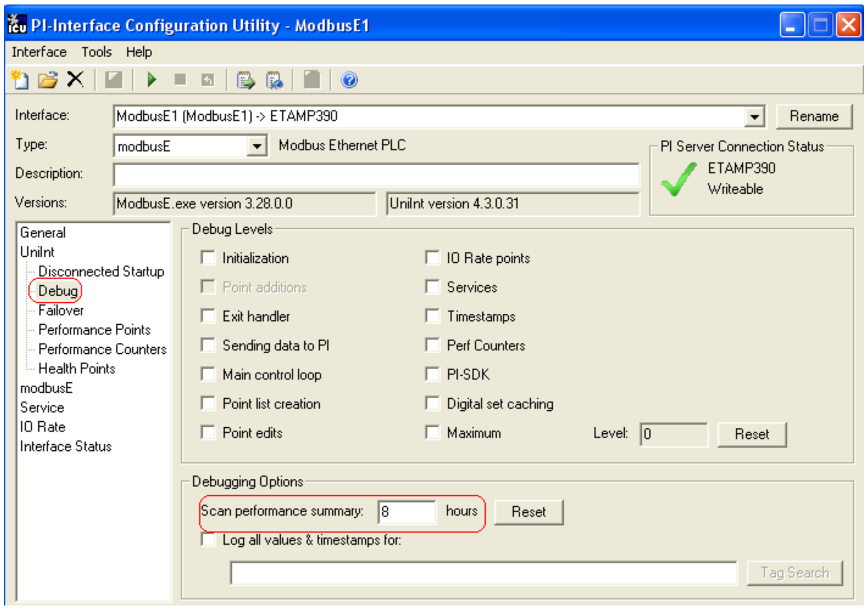
Startup Type: [✓ Auto] Manual Disabled

Chapter 14. Interface Diagnostics Configuration

The Interface Point Configuration chapter provides information on building Historian Points for collecting data from the device. This chapter describes the configuration of points related to interface diagnostics.

Note: The procedure for configuring interface diagnostics is not specific to this Interface. Thus, for simplicity, the instructions and screenshots that follow refer to an interface named **ModbusE**.

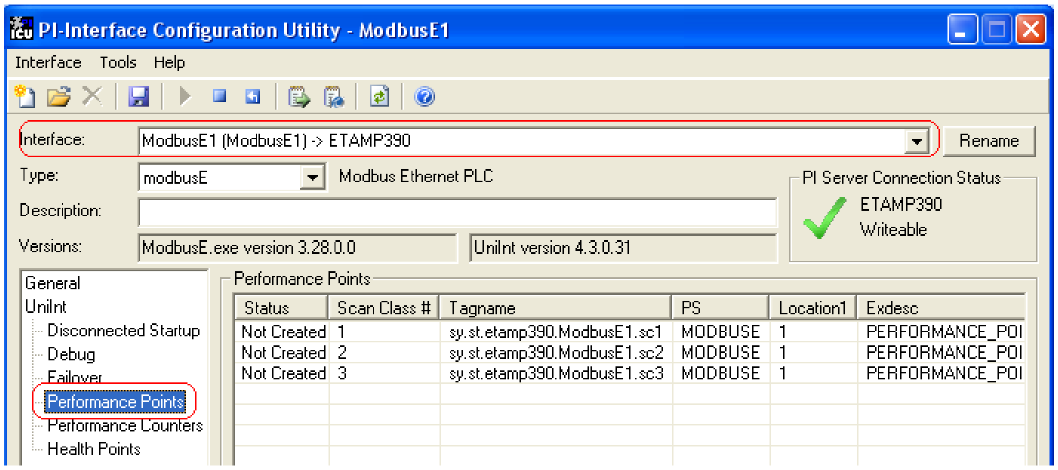
Some of the points that follow refer to a "performance summary interval". This interval is 8 hours by default. You can change this parameter via the *Scan performance summary* box in the *UniInt* - *Debug* parameter category page:



Scan Class Performance Points

A Scan Class Performance Point measures the amount of time (in seconds) that this Interface takes to complete a scan. The Interface writes this scan completion time to millisecond resolution. Scan completion times close to 0 indicate that the interface is performing optimally. Conversely, long scan completion times indicate an increased risk of missed or skipped scans. To prevent missed or skipped scans, you should distribute the data collection points among several scan classes.

You configure one Scan Class Performance Point for each Scan Class in this Interface. From the ICU, select this Interface from the *Interface* drop-down list and click *UniInt-Performance Points* in the parameter category pane:


PI-Interface Configuration Utility - ModbusE1

Interface: ModbusE1 (ModbusE1) -> ETAMP390

Type: modbusE   Modbus Ethernet PLC

Description:

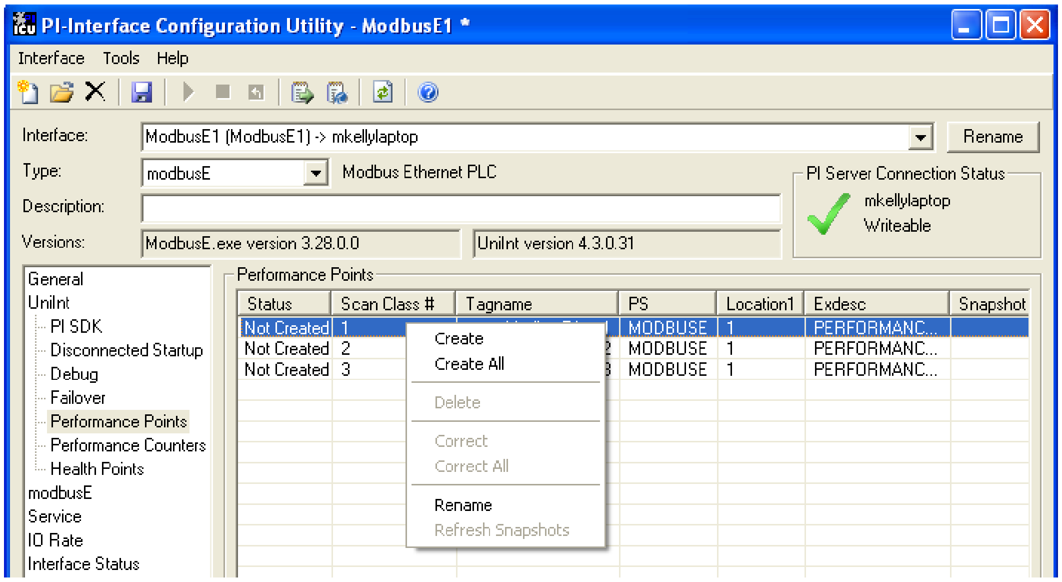
Versions: ModbusE.exe version 3.28.0.0   UniInt version 4.3.0.31

PI Server Connection Status: ETAMP390 Writeable

Performance Points

Status	Scan Class #	Tagname	PS	Location1	Exdesc
Not Created	1	sy.st.etamp390.ModbusE1.sc1	MODBUSE	1	PERFORMANCE_POI
Not Created	2	sy.st.etamp390.ModbusE1.sc2	MODBUSE	1	PERFORMANCE_POI
Not Created	3	sy.st.etamp390.ModbusE1.sc3	MODBUSE	1	PERFORMANCE_POI

Right-click the row for a particular *Scan Class #* to bring up the context menu:


PI-Interface Configuration Utility - ModbusE1 *

Interface: ModbusE1 (ModbusE1) -> mkellylaptop

Type: modbusE   Modbus Ethernet PLC

Description:

Versions: ModbusE.exe version 3.28.0.0   UniInt version 4.3.0.31

PI Server Connection Status: mkellylaptop Writeable

Performance Points

Status	Scan Class #	Tagname	PS	Location1	Exdesc	Snapshot
Not Created	1		MODBUSE	1	PERFORMANC...	
Not Created	2		MODBUSE	1	PERFORMANC...	
Not Created	3		MODBUSE	1	PERFORMANC...	

You need not restart the interface for it to write values to the Scan Class Performance Points.

To see the current values (snapshots) of the Scan Class Performance Points, right-click and select *Refresh Snapshots*.

Create / Create ALL

To create a Performance Point, right-click the line belonging to the tag to be created, and select *Create*. Click *Create All* to create all the Scan Class Performance Points.

Delete

To delete a Performance Point, right-click the line belonging to the tag to be deleted, and select *Delete*.

Correct / Correct All

If the "Status" of a point is marked "Incorrect", the point configuration can be automatically corrected by ICU by right-clicking on the line belonging to the tag to be corrected, and selecting *Correct*. The Performance Points are created with the following Historian attribute values. If ICU detects that a Performance Point is not defined with the following, it will be marked *Incorrect*: To correct all points click the *Correct All* menu item.

The Performance Points are created with the following Historian attribute values:

Attribute	Details
Tag	Tag name that appears in the list box
PointSource	Point Source for tags for this interface, as specified on the first tab
Compressing	Off
ExcMax	0
Descriptor	*Interface name* + " Scan Class # Performance Point"

Rename

Right-click the line belonging to the tag and select *Rename* to rename the Performance Point.

Column descriptions

Status

The *Status* column in the Performance Points table indicates whether the Performance Point exists for the scan class in column 2.

Created - Indicates that the Performance Point does exist

Not Created - Indicates that the Performance Point does not exist

Deleted - Indicates that a Performance Point existed, but was just deleted by the user

Scan Class

The *Scan Class* column indicates which scan class the Performance Point in the *Tagname* column belongs to. There will be one scan class in the *Scan Class* column for each scan class listed in the *Scan Classes* box on the *General* page.

Tagname

The *Tagname* column holds the Performance Point tag name.

PS

This is the point source used for these performance points and the interface.

Location1

This is the value used by the interface for the **`/ID=#`** point attribute.

Exdesc

This is the used to tell the interface that these are performance points and the value is used to corresponds to the **`/ID=#`** command line parameter if multiple copies of the same interface are running on the interface node.

Snapshot

The *Snapshot* column holds the snapshot value of each Performance Point that exists in Historian. The *Snapshot* column is updated when the *Performance Points/Counters* tab is clicked, and when the interface is first loaded. You may have to scroll to the right to see the snapshots.

Performance Counters Points

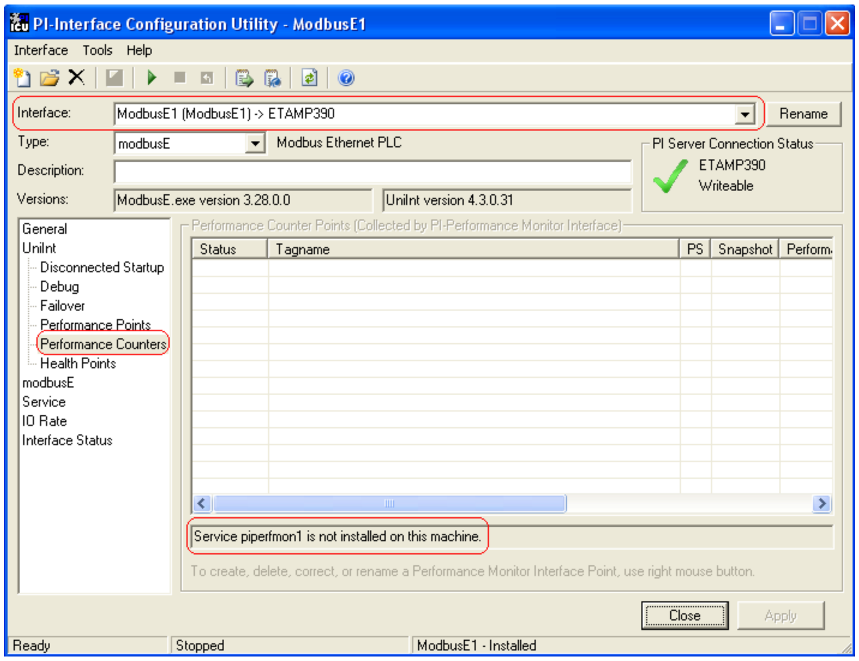
When running as a Service or interactively, this Interface exposes performance data via Windows Performance Counters. Such data include items like:

- the amount of time that the interface has been running;
- the number of points the interface has added to its point list;
- the number of tags that are currently updating among others

There are two types or instances of Performance Counters that can be collected and stored in Historian Points. The first is (_Total) which is a total for the Performance Counter since the interface instance was started. The other is for individual Scan Classes (Scan Class x) where x is a particular scan class defined for the interface instance that is being monitored.

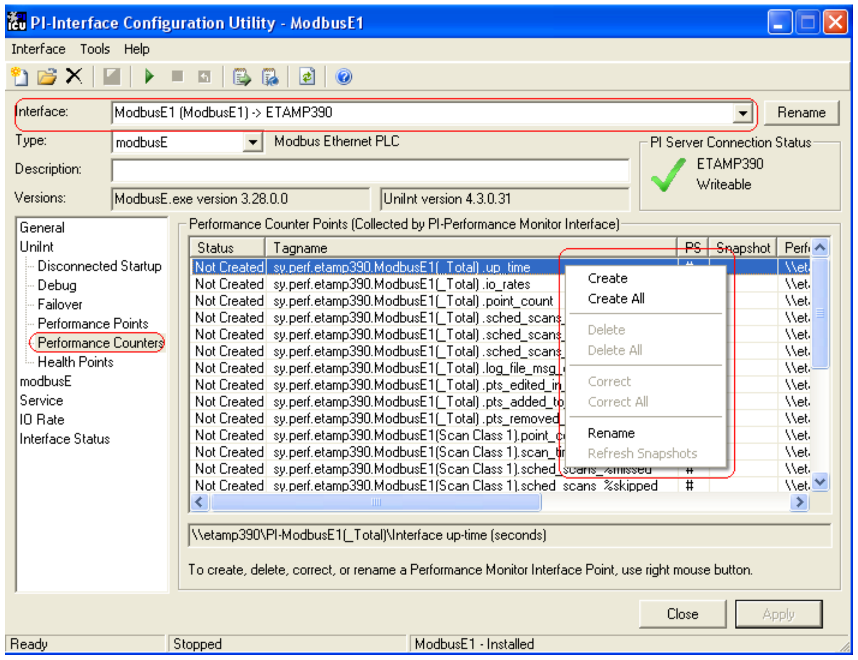
Rockwell Automation's Historian Performance Monitor Interface is capable of reading these performance values and writing them to Historian Points. Please see the *Performance Monitor Interface* for more information.

If there is no Historian Performance Monitor Interface registered with the ICU in the Module Database for the Historian Server the interface is sending its data to, you cannot use the ICU to create any Interface instance's Performance Counters Points:



Interface: ModbusE1 (ModbusE1) -> ETAMP390
Type: modbusE   Modbus Ethernet PLC
Description:
Versions: ModbusE.exe version 3.28.0.0   UniInt version 4.3.0.31
PI Server Connection Status: ETAMP390 Writeable
Performance Counter Points (Collected by PI-Performance Monitor Interface)
Service piperfmon1 is not installed on this machine.
To create, delete, correct, or rename a Performance Monitor Interface Point, use right mouse button.
Ready | Stopped | ModbusE1 - Installed

After installing the Historian Performance Monitor Interface as a service, select this Interface instance from the *Interface* drop-down list, then click *Performance Counters* in the parameter categories pane, and right-click on the row containing the Performance Counters Point you wish to create. This will bring up the context menu:



Interface: ModbusE1 (ModbusE1) -> ETAMP390

Type: modbusE Modbus Ethernet PLC

Description:

Versions: ModbusE.exe version 3.28.0.0 UniInt version 4.3.0.31

PI Server Connection Status: ETAMP390 Writeable

Performance Counter Points (Collected by PI-Performance Monitor Interface)

Status	Tagname	PS	Snapshot	Perf
Not Created	sy.perf.etamp390.ModbusE1(_Total).up_time	#		\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).io_rates			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).point_count			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).sched_scans			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).sched_scans			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).sched_scans			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).log_file_msg			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).pts_edited_in			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).pts_added_to			\\et
Not Created	sy.perf.etamp390.ModbusE1(_Total).pts_removed			\\et
Not Created	sy.perf.etamp390.ModbusE1(Scan Class 1).point_c			\\et
Not Created	sy.perf.etamp390.ModbusE1(Scan Class 1).scan_ti			\\et
Not Created	sy.perf.etamp390.ModbusE1(Scan Class 1).sched_scans_%missed	#		\\et
Not Created	sy.perf.etamp390.ModbusE1(Scan Class 1).sched_scans_%skipped	#		\\et

\\etamp390\PI-ModbusE1(_Total)\Interface up-time (seconds)

To create, delete, correct, or rename a Performance Monitor Interface Point, use right mouse button.

Ready | Stopped | ModbusE1 - Installed

Click *Create* to create the Performance Counters Point for that particular row. Click *Create All* to create all the Performance Counters Points listed which have a status of Not Created.

To see the current values (snapshots) of the created Performance Counters Points, right-click on any row and select *Refresh Snapshots*.

Note: The Historian Performance Monitor Interface - and not this Interface - is responsible for updating the values for the Performance Counters Points in Historian. So, make sure that the Historian Performance Monitor Interface is running correctly.

Performance Counters

In the following lists of Performance Counters the naming convention used will be:

"PerformanceCounterName" (.PerformanceCountersPoint Suffix)

The tagname created by the ICU for each Performance Counter point is based on the setting found under the Tools ➔ Options ➔ Naming Conventions ➔ Performance Counter Points. The default for this is "sy.perf.[machine].[if service]" followed by the Performance Counter Point suffix.

Performance Counters for both (_Total) and (Scan Class x)

"Point Count" (.point_count)

A *.point_count* Performance Counters Point is available for each Scan Class of this Interface as well as a Total for the interface instance.

The *.point_count* Performance Counters Point indicates the number of Historian Points per Scan Class or the total number for the interface instance. This point is similar to the Health Point [UI_SCPOINTCOUNT] for scan classes and [UI_POINTCOUNT] for totals.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).point_count`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on. The tag containing "(_Total)" refers to the sum of all Scan Classes.

"Scheduled Scans: % Missed" (.sched_scans_%missed)

A *.sched_scans_%missed* Performance Counters Point is available for each Scan Class of this Interface as well as a Total for the interface instance.

The *.sched_scans_%missed* Performance Counters Point indicates the percentage of scans the interface missed per Scan Class or the total number missed for all scan classes since startup. A missed scan occurs if the interface performs the scan one second later than scheduled.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).sched_scans_%missed`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on. The tag containing "(_Total)" refers to the sum of all Scan Classes.

"Scheduled Scans: % Skipped" (.sched_scans_%skipped)

A *.sched_scans_%skipped* Performance Counters Point is available for each Scan Class of this Interface as well as a Total for the interface instance.

The *.sched_scans_%skipped* Performance Counters Point indicates the percentage of scans the interface skipped per Scan Class or the total number skipped for all scan classes since startup. A skipped scan is a scan that occurs at least one scan period after its scheduled time. This point is similar to the [UI_SCSKIPPED] Health Point.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).sched_scans_%skipped`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on. The tag containing "(_Total)" refers to the sum of all Scan Classes.

"Scheduled Scans: Scan count this interval" (.sched_scans_this_interval)

A *.sched_scans_this_interval* Performance Counters Point is available for each Scan Class of this Interface as well as a Total for the interface instance.

The *.sched_scans_this_interval* Performance Counters Point indicates the number of scans that the interface performed per performance summary interval for the scan class or the total number of scans performed for all scan classes during the summary interval. This point is similar to the [UI_SCSCANCOUNT] Health Point.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).sched_scans_this_interval`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on. The tag containing "(_Total)" refers to the sum of all Scan Classes.

Performance Counters for (_Total) only

"Device Actual Connections" (.Device_Actual_Connections)

The *.Device_Actual_Connections* Performance Counters Point stores the actual number of foreign devices currently connected and working properly out of the expected number of foreign device connections to the interface. This value will always be less than or equal to the Expected Connections.

"Device Expected Connections" (.Device_Expected_Connections)

The *.Device_Expected_Connections* Performance Counters Point stores the total number of foreign device connections for the interface. This is the expected number of foreign device connections configured that should be working properly at runtime. If the interface can only communicate with 1 foreign device then the value of this counter will always be one. If the interface can support multiple foreign device connections then this is the total number of expected working connections configured for this Interface.

"Device Status" (.Device_Status)

The *.Device_Status* Performance Counters Point stores communication information about the interface and the connection to the foreign device(s). The value of this counter is based on the expected connections, actual connections and value of the **`/PercentUp`** command line option. If the device status is good then the value is '0'. If the device status is bad then the value is '1'. If the interface only supports connecting to 1 foreign device then the **`/PercentUp`** command line value does not change the results of the calculation. If for example the interface can connect to 10 devices and 5 are currently working then the value of the **`/PercentUp`** command line parameter is applied to determine the Device Status. If the value of the **`/PercentUp`** command line parameter is set to 50 and at least 5 devices are working then the DeviceStatus will remain good (i.e. have a value of zero).

"Failover Status" (.Failover_Status)

The *.Failover_Status* Performance Counters Point stores the failover state of the interface when configured for UniInt interface level failover. The value of the counter will be '0' when the interface is running as the 'Primary' interface in the failover configuration. If the interface is running in backup mode then the value of the counter will be '1'.

"Interface up-time (seconds)" (.up_time)

The *.up_time* Performance Counters Point indicates the amount of time (in seconds) that this Interface has been running. At startup the value of the counter is zero. The value will continue to increment until it reaches the maximum value for an unsigned integer. Once it reaches this value then it will start back over at zero.

"IO Rate (events/second)" (.io_rates)

The *.io_rates* Performance Counters Point indicates the rate (in event per second) at which this Interface writes data to its input tags. (As of UniInt 4.5.0.x and later this performance counters point will no longer be available.)

"Log file message count" (.log_file_msg_count)

The *.log_file_msg_count* Performance Counters Point indicates the number of messages that the interface has written to `the log file`. This point is similar to the [UI_MSGCOUNT] Health Point.

"Historian Status" (PI_Status)

The *.PI_Status* Performance Counters Point stores communication information about the interface and the connection to the Historian Server. If the interface is properly communicating with the Historian Server then the value of the counter is '0'. If the communication to the Historian Server goes down for any reason then the value of the counter will be '1'. Once the interface is properly communicating with the Historian Server again then the value will change back to '0'.

"Points added to the interface" (.pts_added_to_interface)

The *.pts_added_to_interface* Performance Counter Point indicates the number of points the interface has added to its point list. This does not include the number of points configured at startup. This is the number of points added to the interface after the interface has finished a successful startup.

"Points edited in the interface"(.pts_edited_in_interface)

The *.pts_edited_in_interface* Performance Counters Point indicates the number of point edits the interface has detected. The Interface detects edits for those points whose `PointSource` attribute matches the Point Source (**`/ps=`**) parameter and whose `Location1` attribute matches the interface ID (**`/id=`**) parameter of the interface.

"Points Good" (.Points_Good)

The *.Points_Good* Performance Counters Point is the number of points that have sent a good current value to Historian. A good value is defined as any value that is not a system digital state value. A point can either be Good, In Error or Stale. The total of Points Good, Points In Error and Points State will equal the Point Count. There is one exception to this rule. At startup of an interface, the Stale timeout must elapse before the point will be added to the Stale Counter. Therefore the interface must be up and running for at least 10 minutes for all tags to belong to a particular Counter.

"Points In Error" (.Points_In_Error)

The *.Points_In_Error* Performance Counters Point indicates the number of points that have sent a current value to Historian that is a system digital state value. Once a point is in the In Error count it will remain in the In Error count until the point receives a new, good value. Points in Error do not transition to the Stale Counter. Only good points become stale.

"Points removed from the interface" (.pts_removed_from_interface)

The *.pts_removed_from_interface* Performance Counters Point indicates the number of points that have been removed from the interface configuration. A point can be removed from the interface when one of the tag properties for the interface is updated and the point is no longer a part of the interface configuration. For example, changing the PointSource, Location1, or Scan attribute can cause the tag to no longer be a part of the interface configuration.

"Points Stale 10(min)" (.Points_Stale_10min)

The *.Points_Stale_10min* Performance Counters Point indicates the number of good points that have not received a new value in the last 10 minutes. If a point is Good, then it will remain in the good list until the Stale timeout elapses. At this time if the point has not received a new value within the Stale Period then the point will move from the Good count to the Stale count. Only points that are Good can become Stale. If the point is in the In Error count then it will remain in the In Error count until the error clears. As stated above, the total count of Points Good, Points In Error and Points Stale will match the Point Count for the interface.

"Points Stale 30(min)" (.Points_Stale_30min)

The *.Points_Stale_30min* Performance Counters Point indicates the number of points that have not received a new value in the last 30 minutes. For a point to be in the Stale 30 minute count it must also be a part of the Stale 10 minute count.

"Points Stale 60(min)" (.Points_Stale_60min)

The *.Points_Stale_30min* Performance Counters Point indicates the number of points that have not received a new value in the last 60 minutes. For a point to be in the Stale 60 minute count it must also be a part of the Stale 10 minute and 30 minute count.

"Points Stale 240(min)" (.Points_Stale_240min)

The *.Points_Stale_240min* Performance Counters Point indicates the number of points that have not received a new value in the last 240 minutes. For a point to be in the Stale 240 minute count it must also be a part of the Stale 10 minute, 30 minute and 60 minute count.

Performance Counters for (Scan Class x) only

"Device Scan Time (milliseconds)" (.Device_Scan_Time)

A *.Device_Scan_Time* Performance Counter Point is available for each Scan Class of this Interface.

The *.Device_Scan_Time* Performance Counters Point indicates the number of milliseconds the interface takes to read the data from the foreign device and package the data to send to Historian. This counter does not include the amount of time to send the data to Historian. This point is similar to the [UI_SCINDEVSCANTIME] Health Point.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1 (Scan Class 1).device_scan _time`" refers to Scan Class 1, "(Scan Class 2) refers to Scan Class 2, and so on.

"Scan Time (milliseconds)" (.scan_time)

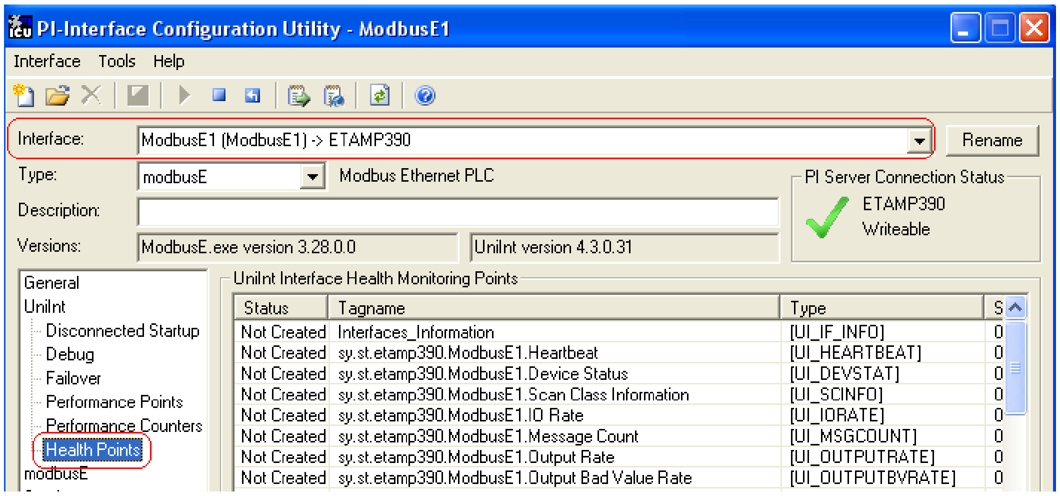
A *.scan_time* Performance Counter Point is available for each Scan Class of this Interface.

The .*scan_time* Performance Counter Point indicates the number of milliseconds the interface takes to both read the data from the device and send the data to Historian. This point is similar to the [UI_SCINSCANTIME] Health Point.

The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).scan_time`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on.

Interface Health Monitoring Points

Interface Health Monitoring Points provide information about the health of this Interface. To use the ICU to configure these points, select this Interface from the *Interface* drop-down list and click *Health Points* from the parameter category pane:

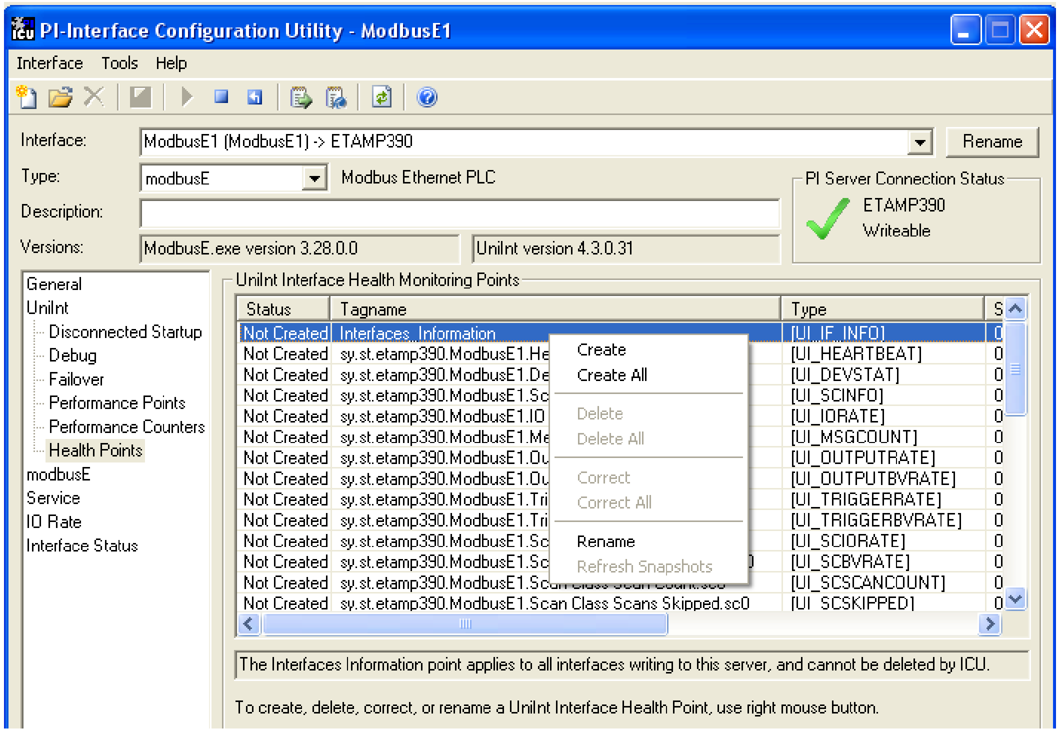

PI-Interface Configuration Utility - ModbusE1

Interface: ModbusE1 (ModbusE1) -> ETAMP390
Type: modbusE Modbus Ethernet PLC
Description:
Versions: ModbusE.exe version 3.28.0.0 | UniInt version 4.3.0.31
PI Server Connection Status: ETAMP390 Writeable

UniInt Interface Health Monitoring Points

Status	Tagname	Type	S
Not Created	Interfaces_Information	[UI_IF_INFO]	0
Not Created	sy.st.etamp390.ModbusE1.Heartbeat	[UI_HEARTBEAT]	0
Not Created	sy.st.etamp390.ModbusE1.Device Status	[UI_DEVSTAT]	0
Not Created	sy.st.etamp390.ModbusE1.Scan Class Information	[UI_SCINFO]	0
Not Created	sy.st.etamp390.ModbusE1.IO Rate	[UI_IORATE]	0
Not Created	sy.st.etamp390.ModbusE1.Message Count	[UI_MSGCOUNT]	0
Not Created	sy.st.etamp390.ModbusE1.Output Rate	[UI_OUTPUTRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Output Bad Value Rate	[UI_OUTPUTBVRATE]	0

Right-click the row for a particular Health Point to display the context menu:


PI-Interface Configuration Utility - ModbusE1

Interface: ModbusE1 (ModbusE1) -> ETAMP390
Type: modbusE Modbus Ethernet PLC
Description:
Versions: ModbusE.exe version 3.28.0.0 | UniInt version 4.3.0.31
PI Server Connection Status: ETAMP390 Writeable

UniInt Interface Health Monitoring Points

Status	Tagname	Type	S
Not Created	Interfaces_Information	[UI_IF_INFO]	0
Not Created	sy.st.etamp390.ModbusE1.He	[UI_HEARTBEAT]	0
Not Created	sy.st.etamp390.ModbusE1.De	[UI_DEVSTAT]	0
Not Created	sy.st.etamp390.ModbusE1.Sc	[UI_SCINFO]	0
Not Created	sy.st.etamp390.ModbusE1.IO	[UI_IORATE]	0
Not Created	sy.st.etamp390.ModbusE1.Me	[UI_MSGCOUNT]	0
Not Created	sy.st.etamp390.ModbusE1.Ou	[UI_OUTPUTRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Ou	[UI_OUTPUTBVRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Tri	[UI_TRIGGERRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Tri	[UI_TRIGGERBVRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Sc	[UI_SCIORATE]	0
Not Created	sy.st.etamp390.ModbusE1.Sc	[UI_SCBVRATE]	0
Not Created	sy.st.etamp390.ModbusE1.Scan Class Scan Count.sc0	[UI_SCSCANCOUNT]	0
Not Created	sy.st.etamp390.ModbusE1.Scan Class Scans Skipped.sc0	[UI_SCSKIPPED]	0

The Interfaces Information point applies to all interfaces writing to this server, and cannot be deleted by ICU.

To create, delete, correct, or rename a UniInt Interface Health Point, use right mouse button.

Click *Create* to create the Health Point for that particular row. Click *Create All* to create all the Health Points.

To see the current values (snapshots) of the Health Points, right-click and select *Refresh Snapshots*.

For some of the Health Points described subsequently, the interface updates their values at each performance summary interval (typically, 8 hours).

[UI_HEARTBEAT]

The [UI_HEARTBEAT] Health Point indicates whether the interface is currently running. The value of this point is an integer that increments continuously from 1 to 15. After reaching 15, the value resets to 1.

The fastest scan class frequency determines the frequency at which the interface updates this point:

Fastest Scan Frequency	Update frequency
Less than 1 second	1 second
Between 1 and 60 seconds, inclusive	Scan frequency
More than 60 seconds	60 seconds

If the value of the [UI_HEARTBEAT] Health Point is not changing, then this Interface is in an unresponsive state.

[UI_DEVSTAT]

The Historian to Historian Interface is built with UniInt 4.4.4.0. New functionality has been added to support health tags. The Health tag with the point attribute `ExDesc = [UI_DEVSTAT]` represents the status of the source device. The following events can be written into this tag:

- “1 | Starting” - the interface is starting.
- “Good” - the interface is properly communicating and reading data from the server.
- The following event represents a failure to communicate with the server:
    - "3 | 1 device(s) in error | Network communication error to source Historian Server"
    - "3 | 1 device(s) in error | Unable to get archive data from source Historian Server"
    - "3 | 1 device(s) in error | Unable to get snapshot data from source Historian Server"
    - "3 | 1 device(s) in error | Unable to write data to receiving Historian Server"
    - "3 | 1 device(s) in error | Unable to obtain current data with source Historian Server failover enabled."
- “4 | Intf Shutdown” - the interface is stopped.

Refer to the *UniInt Interface User Manual* for more information on how to configure health points.

[UI_SCINFO]

The [UI_SCINFO] Health Point provides scan class information. The value of this point is a string that indicates

- the number of scan classes;
- the update frequency of the [UI_HEARTBEAT] Health Point; and
- the scan class frequencies

An example value for the [UI_SCINFO] Health Point is:

```
3 | 5 | 5 | 60 | 120
```

The Interface updates the value of this point at startup and at each performance summary interval.

[UI_IORATE]

The [UI_IORATE] Health Point indicates the sum of

1. the number of scan-based input values the interface collects before it performs exception reporting; and
2. the number of event-based input values the interface collects before it performs exception reporting; and
3. the number of values that the interface writes to output tags that have a `SourceTag`.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point. The value of this [UI_IORATE] Health Point may be zero. A stale timestamp for this point indicates that this Interface has stopped collecting data.

[UI_MSGCOUNT]

The [UI_MSGCOUNT] Health Point tracks the number of messages that the interface has written to the `pipc.log` file since start-up. In general, a large number for this point indicates that the interface is encountering problems. You should investigate the cause of these problems by looking in `pipc.log`.

The Interface updates the value of this point every 60 seconds. While the interface is running, the value of this point never decreases.

[UI_POINTCOUNT]

The [UI_POINTCOUNT] Health Point counts number of Historian tags loaded by the interface. This count includes all input, output and triggered input tags. This count does NOT include any Interface Health tags or performance points.

The interface updates the value of this point at startup, on change and at shutdown.

[UI_OUTPUTRATE]

After performing an output to the device, this Interface writes the output value to the output tag if the tag has a `SourceTag`. The [UI_OUTPUTRATE] Health Point tracks the number of these values. If there are no output tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point. The Interface resets the value of this point to zero at each performance summary interval.

[UI_OUTPUTBVRATE]

The [UI_OUTPUTBVRATE] Health Point tracks the number of System Digital State values that the interface writes to output tags that have a `SourceTag`. If there are no output tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point. The Interface resets the value of this point to zero at each performance summary interval.

[UI_TRIGGERRATE]

The [UI_TRIGGERRATE] Health Point tracks the number of values that the interface writes to event-based input tags. If there are no event-based input tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point. The Interface resets the value of this point to zero at each performance summary interval.

[UI_TRIGGERBVRATE]

The [UI_TRIGGERRATE] Health Point tracks the number of System Digital State values that the interface writes to event-based input tags. If there are no event-based input tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point. The Interface resets the value of this point to zero at each performance summary interval.

[UI_SCIORATE]

You can create a [UI_SCIORATE] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix “.sc1” (for example, `sy.st.etamp390.E1.Scan Class IO Rate.sc1`) refers to Scan Class 1, “.sc2” refers to Scan Class 2, and so on.

A particular Scan Class’s [UI_SCIORATE] point indicates the number of values that the interface has collected. If the current value of this point is between zero and the corresponding [UI_SCPOINTCOUNT] point, inclusive, then the interface executed the scan successfully. If a [UI_SCIORATE] point stops updating, then this condition indicates that an error has occurred and the tags for the scan class are no longer receiving new data.

The Interface updates the value of a [UI_SCIORATE] point after the completion of the associated scan.

Although the ICU allows you to create the point with the suffix “.sc0”, this point is not applicable to this Interface.

[UI_SCBVRATE]

You can create a [UI_SCBVRATE] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix “.sc1” (for example, `sy.st.etamp390.E1.Scan Class Bad Value Rate.sc1`) refers to Scan Class 1, “.sc2” refers to Scan Class 2, and so on.

A particular Scan Class's [UI_SCBVRATE] point indicates the number System Digital State values that the interface has collected.

The Interface updates the value of a [UI_SCBVRATE] point after the completion of the associated scan.

Although the ICU allows you to create the point with the suffix ".sc0", this point is not applicable to this Interface.

[UI_SCSCANCOUNT]

You can create a [UI_SCSCANCOUNT] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example, `sy.st.etamp390.E1.Scan Class Scan Count.sc1`) refers to Scan Class 1, ".sc2" refers to Scan Class 2, and so on.

A particular Scan Class's [UI_ SCSCANCOUNT] point tracks the number of scans that the interface has performed.

The Interface updates the value of this point at the completion of the associated scan. The Interface resets the value to zero at each performance summary interval.

Although there is no "Scan Class 0", the ICU allows you to create the point with the suffix ".sc0". This point indicates the total number of scans the interface has performed for all of its Scan Classes.

[UI_SCSKIPPED]

You can create a [UI_SCSKIPPED] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example, `sy.st.etamp390.E1.Scan Class Scans Skipped.sc1`) refers to Scan Class 1, ".sc2" refers to Scan Class 2, and so on.

A particular Scan Class's [UI_SCSKIPPED] point tracks the number of scans that the interface was not able to perform before the scan time elapsed and before the interface performed the next scheduled scan.

The Interface updates the value of this point each time it skips a scan. The value represents the total number of skipped scans since the previous performance summary interval. The Interface resets the value of this point to zero at each performance summary interval.

Although there is no "Scan Class 0", the ICU allows you to create the point with the suffix ".sc0". This point monitors the total skipped scans for all of the interface's Scan Classes.

[UI_SCPOINTCOUNT]

You can create a [UI_SCPOINTCOUNT] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example, `sy.st.etamp390.E1.Scan Class Point Count.sc1`) refers to Scan Class 1, ".sc2" refers to Scan Class 2, and so on.

This Health Point monitors the number of tags in a Scan Class.

The Interface updates a [UI_SCPOINTCOUNT] Health Point when it performs the associated scan.

Although the ICU allows you to create the point with the suffix ".sc0", this point is not applicable to this Interface.

[UI_SCINSCANTIME]

You can create a [UI_SCINSCANTIME] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example, `sy.st.etamp390.E1.Scan Class Scan Time.sc1`) refers to Scan Class 1, ".sc2" refers to Scan Class 2, and so on.

A particular Scan Class's [UI_ SCINSCANTIME] point represents the amount of time (in milliseconds) the interface takes to read data from the device, fill in the values for the tags, and send the values to the Historian Server.

The Interface updates the value of this point at the completion of the associated scan.

[UI_SCINDEVSCANTIME]

You can create a [UI_SCINDEVSCANTIME] Health Point for each Scan Class in this Interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example, `sy.st.etamp390.E1.Scan Class Device Scan Time.sc1`) refers to Scan Class 1, ".sc2" refers to Scan Class 2, and so on.

A particular Scan Class's [UI_ SCINDEVSCANTIME] point represents the amount of time (in milliseconds) the interface takes to read data from the device and fill in the values for the tags.

The value of a [UI_ SCINDEVSCANTIME] point is a fraction of the corresponding [UI_SCINSCANTIME] point value. You can use these numbers to determine the percentage of time the interface spends communicating with the device compared with the percentage of time communicating with the Historian Server.

If the [UI_SCSKIPPED] value is increasing, the [UI_SCINDEVSCANTIME] points along with the [UI_SCINSCANTIME] points can help identify where the delay is occurring: whether the reason is communication with the device, communication with the Historian Server, or elsewhere.

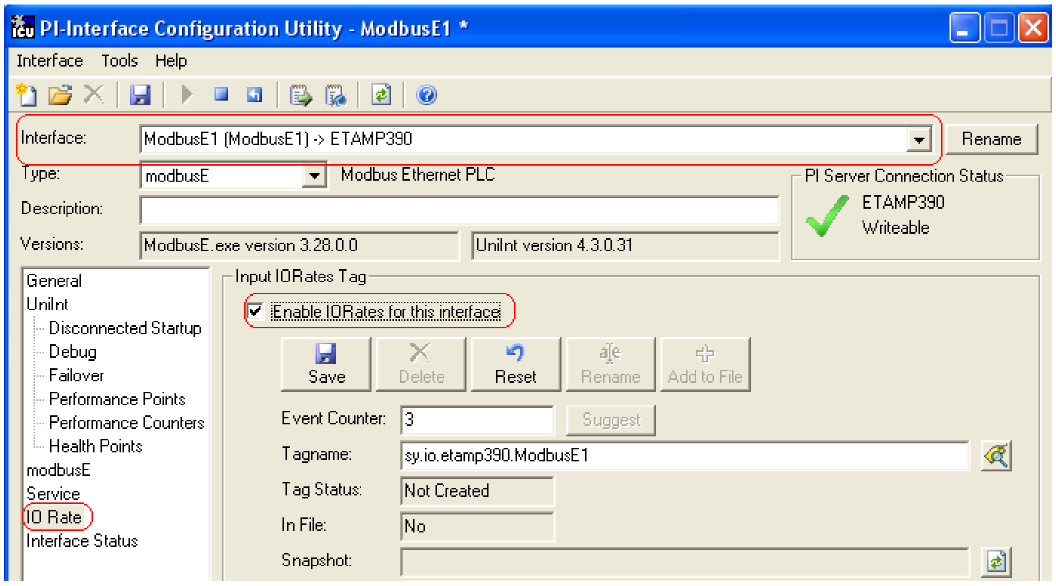
The Interface updates the value of this point at the completion of the associated scan.

I/O Rate Point

An I/O Rate point measures the rate at which the interface writes data to its input tags. The value of an I/O Rate point represents a 10-minute average of the total number of values per minute that the interface sends to the Historian Server.

When the interface starts, it writes 0 to the I/O Rate point. After running for ten minutes, the interface writes the I/O Rate value. The Interface continues to write a value every 10 minutes. When the interface stops, it writes 0.

The ICU allows you to create one I/O Rate point for each copy of this Interface. Select this Interface from the *Interface* drop-down list, click *IO Rate* in the parameter category pane, and check *Enable IORates for this Interface*.

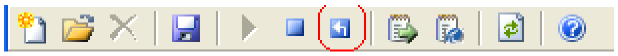


Interface: ModbusE1 (ModbusE1) -> ETAMP390
Type: modbusE Modbus Ethernet PLC
Description:
Versions: ModbusE.exe version 3.28.0.0 UniInt version 4.3.0.31

[✓] Enable IORates for this interface
Event Counter: 3
Tagname: sy.io.etamp390.ModbusE1
Tag Status: Not Created
In File: No
Snapshot:

As the preceding picture shows, the ICU suggests an *Event Counter* number and a *Tagname* for the I/O Rate Point. Click the *Save* button to save the settings and create the I/O Rate point. Click the *Apply* button to apply the changes to this copy of the interface.

You need to restart the interface in order for it to write a value to the newly created I/O Rate point. Restart the interface by clicking the *Restart* button:

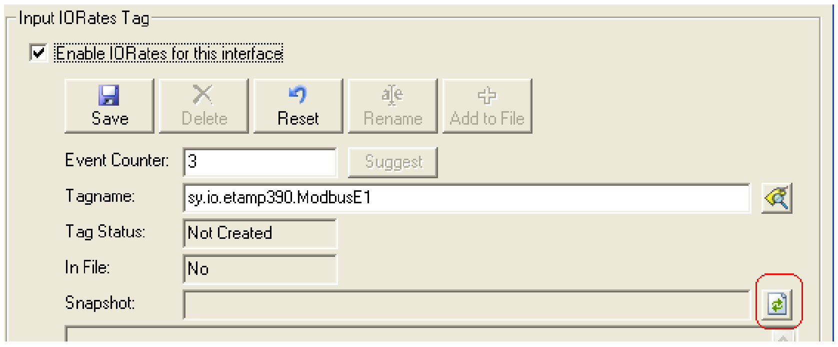


(The reason you need to restart the interface is that the `PointSource` attribute of an I/O Rate point is `Lab`.)

To confirm that the interface recognizes the I/O Rate Point, look in the `pipc.log` for a message such as:

```
Historian-ModBus 1> IORATE: tag sy.io.etamp390.ModbusE1 configured.
```

To see the I/O Rate point's current value (snapshot), click the *Refresh snapshot* button:



[✓] Enable IORates for this interface
Event Counter: 3
Tagname: sy.io.etamp390.ModbusE1
Tag Status: Not Created
In File: No
Snapshot:

Enable IORates for this Interface

The *Enable IORates for this interface* check box enables or disables I/O Rates for the current interface. To disable I/O Rates for the selected interface, uncheck this box. To enable I/O Rates for the selected interface, check this box.

Event Counter

The *Event Counter* correlates a tag specified in the iorates.dat file with this copy of the interface. The command-line equivalent is **`/ec=x`**, where `x` is the same number that is assigned to a tag name in the iorates.dat file.

Tagname

The tag name listed under the *Tagname* column is the name of the I/O Rate tag.

Tag Status

The *Tag Status* column indicates whether the I/O Rate tag exists in Historian. The possible states are:

- Created - This status indicates that the tag exist in Historian
- Not Created - This status indicates that the tag does not yet exist in Historian
- Deleted - This status indicates that the tag has just been deleted
- Unknown - This status indicates that the ICU is not able to access the Historian Server

In File

The *In File* column indicates whether the I/O Rate tag listed in the tag name and the event counter is in the IORates.dat file. The possible states are:

- Yes - This status indicates that the tag name and event counter are in the IORates.dat file
- No - This status indicates that the tag name and event counter are not in the IORates.dat file

Snapshot

The *Snapshot* column holds the snapshot value of the I/O Rate tag, if the I/O Rate tag exists in Historian. The *Snapshot* column is updated when the *IORates/Status Tags* tab is clicked, and when the interface is first loaded.

Right Mouse Button Menu Options

Create

Create the suggested I/O Rate tag with the tag name indicated in the *Tagname* column.

Delete

Delete the I/O Rate tag listed in the *Tagname* column.

Rename

Allow the user to specify a new name for the I/O Rate tag listed in the *Tagname* column.

Add to File

Add the tag to the IORates.dat file with the event counter listed in the *Event Counter* Column.

Search

Allow the user to search the Historian Server for a previously defined I/O Rate tag.

Interface Status Point

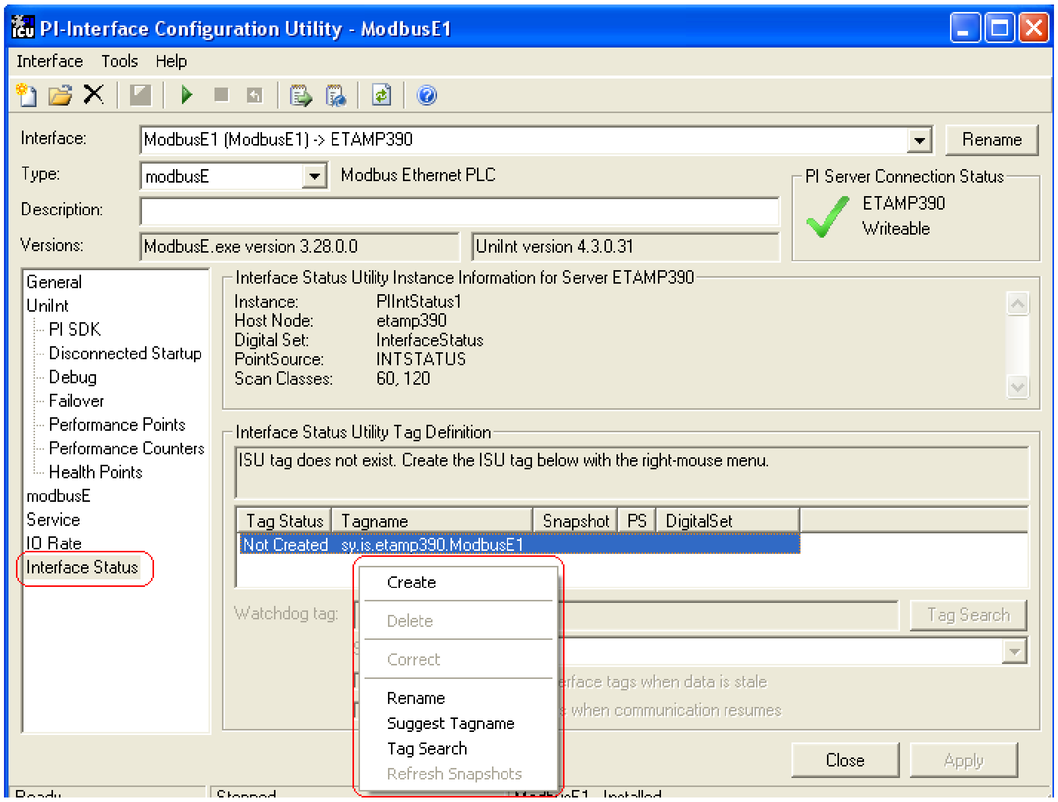
The Historian Interface Status Utility (ISU) alerts you when an interface is not currently writing data to the Historian Server. This situation commonly occurs if

- the monitored interface is running on an interface node, but the interface node cannot communicate with the Historian Server; or
- the monitored interface is not running, but it failed to write at shutdown a System state such as `Intf Shut`.

The ISU works by periodically looking at the timestamp of a Watchdog Tag. The Watchdog Tag is a tag whose value a monitored interface (such as this Interface) frequently updates. The Watchdog Tag has its `ExcDev`, `ExcMin`, and `ExcMax` point attributes set to 0. So, a non-changing timestamp for the Watchdog Tag indicates that the monitored interface is not writing data.

Please see the *Interface Status Interface* for complete information on using the ISU. Historian Interface Status Utility runs only on a Historian Server Node.

If you have used the ICU to configure the Historian Interface Status Utility on the Historian Server Node, the ICU allows you to create the appropriate ISU point. Select this Interface from the *Interface* drop-down list and click *Interface Status* in the parameter category pane. Right-click on the ISU tag definition window to bring up the context menu:



Interface: ModbusE1 (ModbusE1) -> ETAMP390
Type: modbusE — Modbus Ethernet PLC
Versions: ModbusE.exe version 3.28.0.0 — UniInt version 4.3.0.31

Instance:	PIIntStatus1
Host Node:	etamp390
Digital Set:	InterfaceStatus
PointSource:	INTSTATUS
Scan Classes:	60, 120

ISU tag does not exist. Create the ISU tag below with the right-mouse menu.

Tag Status	Tagname	Snapshot	PS	DigitalSet
Not Created	sy.is.etamp390.ModbusE1			

…rface tags when data is stale
… when communication resumes

Click *Create* to create the ISU tag.

Use the *Tag Search* button to select a Watchdog Tag. (Recall that the Watchdog Tag is one of the points for which this Interface collects data.)

Select a *Scan frequency* from the drop-down list box. This Scan frequency is the interval at which the ISU monitors the Watchdog Tag. For optimal performance, choose a *Scan frequency* that is less frequent than the majority of the scan rates for this Interface's points. For example, if this Interface scans most of its points every 30 seconds, choose a *Scan frequency* of 60 seconds. If this Interface scans most of its points every second, choose a *Scan frequency* of 10 seconds.

If the *Tag Status* indicates that the ISU tag is `Incorrect`, right-click to enable the context menu and select *Correct*.

Note: The Historian Interface Status Utility - and not this Interface - is responsible for updating the ISU tag. So, make sure that the Historian Interface Status Utility is running correctly.

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`** parameter on the startup command-line.

Message Logs

The location of the message log depends upon the platform on which the interface is running. See the *UniInt Interface User Manual* for more information.

Messages are written to *`PIHOME\dat\pipc.log`* at the following times.

- When the interface starts many informational messages are written to the log. These include the version of the interface, the version of UniInt, the command-line parameters used, and the number of points.
- As the interface retrieves points, messages are sent to the log if there are any problems with the configuration of the points.
- If the **`/dbUniInt`** is used on the command-line, then various informational messages are written to the log file.

Messages

Interface Startup Messages

When the interface is started, the log file will contain informational messages that describe the settings that will be used from the startup script and the `PItoPI.ini` file. The interface will then print a message stating the receiving Historian Server and a message for each source Historian Server. The message for each source Historian Server also states the time offset between source and receiving Historian Server along with time zone differences. Note that the `PItoPI.ini` can be used to configure a different source server for each scan class. In addition each source server (and therefore scan class) can have different history recovery parameters. Finally the `PItoPI.ini` can be used to specify which scan class signs up for exceptions (default is scan class 1 gets exception data unless **`/hronly`** is specified in `PItoPI.bat`). After printing a message specifying the receiving and source Historian Servers it prints a message for each scan class stating if it will collect archive or exception data, the source Historian Server for that scan class and the history recovery parameters.

Scan Summary

After a scan class has finished history recovery, the interface logs messages indicating the number of tags in error for the scan class.

```
PItoPI- 1> Scan class 1: History recovery completed successfully.
```

```
PItoPI- 1> Scan class 1: 0 of 385 points in error.
```

The messages above indicate the number of points successfully registered with the update manager on the source node and the number of errors encountered in the given scan class.

System Errors and Historian Errors

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

Error Descriptions

Descriptions of system and Historian errors can be obtained with the `pidiag` utility:

Syntax: `\PI\adm\pidiag -e error_number`

Historian to Historian Specific Error Messages

Message	`16-May-06 17:29:06`
	`PItoPI 1> Error -77 returned from pisn_evmexceptions cll to source Historian Server.`

Cause	Update manager queue limit has been reached on the source Historian Server.
Resolution	Increase PI Update Manager queue size limits on source Historian Server as described in Exception Data Collection section: Historian 3 Update Queue Size Limits.

Message	`16-May-06 17:29:06`
	`PItoPI 1> Tag SINUSOID rejected.Source tag already configured for tag SINUSOID.`

Cause	UniInt tag loading behavior. When UniInt is disconnected from **`/host`** Historian Server on startup it will does not resume tag loading where it left off. Instead it reloads the tag list from the beginning. Any tag that was loaded before the disconnection will be rejected after reconnection with the stated error message. This error message can safely be ignored. The tag is loaded and will receive data.
Resolution	See PLI 19689OSI8. This issue will be addressed in a future UniInt/PItoPI release.

UniInt Failover Specific Error Messages

Informational

Message	`16-May-06 10:38:00` `PItoPI 1> UniInt failover: Interface in the "Backup" state.`
Meaning	Upon system startup, the initial transition is made to this state. While in this state the interface monitors the status of the other interface participating in failover. When configured for Hot failover, data received from the data source is queued and not sent to the Historian Server while in this state. The amount of data queued while in this state is determined by the failover update interval. In any case, there will be typically no more than two update intervals of data in the queue at any given time. Some transition chains may cause the queue to hold up to five failover update intervals worth of data.

Message	`16-May-06 10:38:05` `PItoPI 1> UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface not available.`
Meaning	While in this state, the interface is in its primary role and sends data to the Historian Server as it is received. This message also states that there is not a backup interface participating in failover.

Message	`16-May-06 16:37:21` `PItoPI 1> UniInt failover: Interface in the "Primary" state and actively sending data to Historian. Backup interface available.`
Meaning	While in this state, the interface sends data to the Historian Server as it is received. This message also states that the other copy of the interface appears to be ready to take over the role of primary.

Errors (Phase 1 & 2)

Message	`16-May-06 17:29:06` `PItoPI 1> One of the required Failover Synchronization points was not loaded.` ` Error = 0: The Active ID synchronization point was not loaded.` `The input Historian tag was not loaded`
Cause	The Active ID tag is not configured properly.
Resolution	Check validity of point attributes. For example, make sure Location1 attribute is valid for the interface. All failover tags must have the same `PointSource` and `Location1` attributes. Modify point attributes as necessary and restart the interface.

Message	`16-May-06 17:38:06`
	`PItoPI 1> One of the required Failover Synchronization points was not loaded.`
	`Error = 0: The Heartbeat point for this copy of the interface was not loaded.`
	`The input Historian tag was not loaded`

Cause	The Heartbeat tag is not configured properly.
Resolution	Check validity of point attributes. For example, make sure Location1 attribute is valid for the interface. All failover tags must have the same PointSource and Location1 attributes. Modify point attributes as necessary and restart the interface.

Message	`17-May-06 09:06:03`
	`PItoPI > The Uniint FailOver ID (`**`/UFO_ID`**`) must be a positive integer.`

Cause	The UFO_ID parameter has not been assigned a positive integer value.
Resolution	Change and verify the parameter to a positive integer and restart the interface.

Message	`17-May-06 09:06:03`
	`PItoPI 1> The Failover ID parameter (`**`/UFO_ID`**`) was found but the ID for the redundant copy was not found`

Cause	The **`/UFO_OtherID`** parameter is not defined or has not been assigned a positive integer value.
Resolution	Change and verify the **`/UFO_OtherID`** parameter to a positive integer and restart the interface.

Errors (Phase 2)

Unable to open synchronization file

Message	`27-Jun-08 17:27:17`
	`Historian Eight Track 1 1> Error 5: Unable to create file '\\georgiaking\GeorgiaKingStorage\UnIntFailover\\PIEightTrack_eight_1.dat'`
	`Verify that interface has read/write/create access on file server machine.`
	`Intializing uniint library failed`
	`Stopping Interface`

Cause	This message will be seen when the interface is unable to create a new failover synchronization file at startup. The creation of the file only takes place the first time either copy of the interface is started and the file does not exist. The error number most commonly seen is error number 5. Error number 5 is an “access denied” error and is likely the result of a permissions problem.
Resolution	Ensure the account the interface is running under has read and write permissions for the folder. The “log on as” property of the Windows service may need to be set to an account that has permissions for the folder.

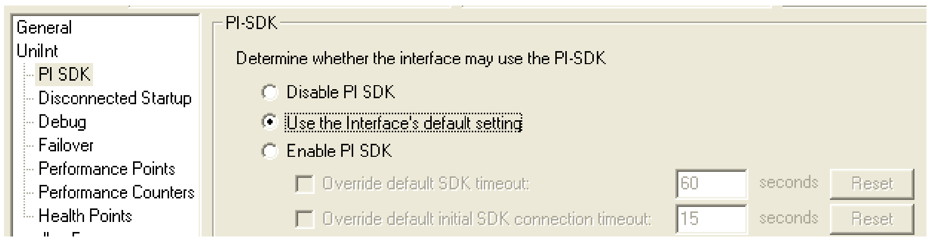
Error Opening Synchronization File

Message	`Sun Jun 29 17:18:51 2008`
	`Historian Eight Track 1 2> WARNING> Failover Warning: Error = 64`
	`Unable to open Failover Control File '\\georgiaking\GeorgiaKingStorage\Eight\PIEightTrack_eight_1.dat'`
	`The interface will not be able to change state if Historian is not available`

Cause	This message will be seen when the interface is unable to open the failover synchronization file. The interface failover will continue to operate correctly as long as communication to the Historian Server is not interrupted. If communication to Historian is interrupted while one or both interfaces cannot access the synchronization file, the interfaces will remain in the state they were in at the time of the second failure, so the primary interface will remain primary and the backup interface will remain backup.
Resolution	Ensure the account the interface is running under has read and write permissions for the folder and file. The “log on as” property of the Windows service may need to be set to an account that has permissions for the folder and file.

Appendix B. PI SDK Options

To access the PI SDK settings for this Interface, select this Interface from the *Interface* drop-down list and click *UniInt - PI SDK* in the parameter category pane.



Determine whether the interface may use the PI-SDK

( ) Disable PI SDK
[✓ Use the Interface's default setting]
( ) Enable PI SDK

☐ Override default SDK timeout: 60 seconds
☐ Override default initial SDK connection timeout: 15 seconds

Disable PI SDK

Select *Disable PI SDK* to tell the interface not to use the PI SDK. If you want to run the interface in Disconnected Startup mode, you must choose this option.

The command line equivalent for this option is **`/pisdk=0`**.

Use the Interface's default setting

This selection has no effect on whether the interface uses the PI SDK. However, you must not choose this option if you want to run the interface in Disconnected Startup mode.

Enable PI SDK

Select *Enable PI SDK* to tell the interface to use the PI SDK. Choose this option if the Historian Server version is earlier than 2.x or the PI API is earlier than 1.6.0.2, and you want to use extended lengths for the `Tag`, `Descriptor`, `ExDesc`, `InstrumentTag`, or `PointSource` point attributes. The maximum lengths for these attributes are:

Attribute	Enable the interface to use the PI SDK	Historian Server earlier than 2.x or PI API earlier than 1.6.0.2, without the use of the PI SDK
Tag	1023	255
Descriptor	1023	26
ExDesc	1023	80
InstrumentTag	1023	32
PointSource	1023	1

However, if you want to run the interface in Disconnected Startup mode, you must not choose this option.

The command line equivalent for this option is **`/pisdk=1`**.

Appendix C. Terminology

To understand this interface manual, you should be familiar with the terminology used in this document.

Buffering

Buffering refers to an interface node's ability to store temporarily the data that interfaces collect and to forward these data to the appropriate Historian Servers.

N-Way Buffering

If you have Historian Servers that are part of a Historian Collective, PIBufss supports n-way buffering. N-way buffering refers to the ability of a buffering application to send the same data to each of the Historian Servers in a Historian Collective. (Bufserv also supports n-way buffering to multiple Historian Server however it does not guarantee identical archive records since point compressions specs could be different between Historian Servers. With this in mind, Rockwell Automation recommends that you run PIBufss instead.)

ICU

ICU refers to the Historian Interface Configuration Utility. The ICU is the primary application that you use to configure Historian interface programs. You must install the ICU on the same computer on which an interface runs. A single copy of the ICU manages all of the interfaces on a particular computer.

You can configure an interface by editing a startup command file. However, Rockwell Automation discourages this approach. Instead, Rockwell Automation strongly recommends that you use the ICU for interface management tasks.

ICU Control

An ICU Control is a plug-in to the ICU. Whereas the ICU handles functionality common to all interfaces, an ICU Control implements interface-specific behavior. Most Historian Interfaces have an associated ICU Control.

Interface Node

An interface node is a computer on which

- the PI API and/or PI SDK are installed, and
- Historian Server programs are not installed.

PI API

The PI API is a library of functions that allow applications to communicate and exchange data with the Historian Server. All Historian Interfaces use the PI API.

Historian Collective

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

PIHOME

`PIHOME` refers to the directory that is the common location for Historian 32-bit client applications.

On a 32-bit operating system

A typical `PIHOME` is `C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC`.

On a 64-bit operating system

A typical `PIHOME` is `C:\Program Files (x86)\PIPC.`

Historian 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

`PIHOME64` will be found only on a 64-bit operating system and refers to the directory that is the common location for Historian 64-bit client applications.

A typical `PIHOME64` is `C:\Program Files\Rockwell Software\FactoryTalk Historian\PIPC`.

Historian Interfaces reside in a subdirectory of the `Interfaces` directory under `PIHOME64`.

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

`C:\Program Files\Rockwell Software\FactoryTalk Historian\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

The PI SDK is a library of functions that allow applications to communicate and exchange data with the Historian Server. Some Historian Interfaces, in addition to using the PI API, require the use of the PI SDK.

Historian Server Node

A Historian Server Node is a computer on which Historian Server programs are installed. The Historian Server runs on the Historian Server Node.

SMT

SMT refers to System Management Tools. SMT is the program that you use for configuring Historian Servers. A single copy of SMT manages multiple Historian Servers. SMT runs on either a Historian Server Node or an interface node.

pipc.log

The `pipc.log` file is the file to which OSIsoft applications write informational and error messages. When a Historian interface runs, it writes to the `pipc.log` file. The ICU allows easy access to the `pipc.log`.

Point

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

A PI point does not necessarily correspond to a “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 Historian Points.

Service

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

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

Tag (Input Tag and Output Tag)

The tag attribute of a PI point is 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, FactoryTalk Historian 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.

Appendix D. Technical Support and Resources

Rockwell provides dedicated technical support internationally, 24 hours a day, 7 days a week.

You can read complete information about technical support options, and access all of the following resources at the *Rockwell Automation Support Web site* (*http://www.rockwellautomation.com/support/*).

Technical Support

Please visit *Rockwell Automation Customer Support Center* (*http://www.rockwellautomation.com/support/*) for access to user forums, sample code, software and firmware updates, product manuals, and other downloads.

Knowledgebase

The Customer Support Center offers an extensive online knowledgebase that includes frequently asked questions (FAQs) and the latest patches. Please visit the *support site* (*http://www.rockwellautomation.com/resources/support.html*) and select the **Knowledgebase** link located under **Tools & Resources** to:

- View technical and application notes.
- Obtain software patches and firmware updates.
- Subscribe to product and service e-mail notifications.
- Ask questions.

Worldwide Support

If you are not located in North America and want to contact Rockwell Automation Support, use the *Worldwide Locator* (*http://www.rockwellautomation.com/locations/*) for worldwide contact information.

Training Programs

Rockwell Automation offers a wide range of training programs that include e-learning, regularly scheduled and custom-tailored classes, self-paced training, and certificate programs. If you would like more information on training, visit the *Rockwell Automation Training site* (*http://www.rockwellautomation.com/services/training/*) or call 1.440.646.3434.

If you are not located in North America and want to contact Rockwell Automation Support, use the *Worldwide Locator* (*http://www.rockwellautomation.com/locations/*) for worldwide contact information.

Consulting Services

If you are not located in North America and want to contact Rockwell Automation Support, use the *Worldwide Locator* (*http://www.rockwellautomation.com/locations/*) for worldwide contact information.

TechConnect Support

With TechConnect Support, your site has unlimited, real-time access to Rockwell Automation's global network of Customer Support Centers and technical resources. TechConnect service levels are provided at the *TechConnect site* (*http://www.rockwellautomation.com/services/onlinephone/techconnect/*).

When you contact Rockwell Technical Support, please provide:

- Product name, version, and/or build numbers.
- Computer platform (CPU type, operating system, and version number).
- Exact wording of any messages that appeared on your screen.
- The message log(s) at that time.
- Descriptions of:
    - What happened and what you were doing when the problem occurred.
    - How you tried to solve the problem.

Find the Version and Build Numbers

To find version and build numbers for each Historian Server subsystem (which vary depending on installed upgrades, updates or patches), use either of the following methods:

To check the numbers with System Management Tools (SMT):

1. Go to **Start > All Programs > Rockwell Software > FactoryTalk Historian SE > System Management Tools**. The **<STM>** dialog box appears.
1. Under **Collectives and Servers**, select the name of the server you want to check.
2. Under **System Management Tools**, select **Operation > PI Version**.

   The **Version in Memory** and **Version on Disk** columns display information on versions of all the server subsystems.

View Computer Platform Information

To view platform specifications, right-click **My Computer** and select **Properties**. For more detailed information, choose **Start > Run**, and type *msinfo32.exe*.