PI Interface for GE iFix

Version 2.6.2.x

OSIsoft, LLC
777 Davis St., Suite 250
San Leandro, CA 94577 USA
Tel: (01) 510-297-5800
Fax: (01) 510-357-8136
Web: http://www.osisoft.com

OSIsoft Australia • Perth, Australia
OSIsoft Europe GmbH • Frankfurt, Germany
OSIsoft Asia Pte Ltd. • Singapore
OSIsoft Canada ULC • Montreal & Calgary, Canada
OSIsoft, LLC Representative Office • Shanghai, People's Republic of China
OSIsoft Japan KK • Tokyo, Japan
OSIsoft Mexico S. De R.L. De C.V. • Mexico City, Mexico
OSIsoft do Brasil Sistemas Ltda. • Sao Paulo, Brazil
OSIsoft France EURL • Paris, France

Published: 11/2014

Table of Contents

Chapter 1. Introduction

The PI IntFix Interface moves data between Intellution FIX/iFIX software platforms and the PI Data Archive. The interface program reads the PI point database to determine which points to read. It then queries the local Intellution node for current values and sends exception reports to the PI system. The interface can also write values back to the local Intellution database(s).

The interface runs on Windows platforms. It communicates using Intellution's EDA (Easy Data Access) library and can be run on either a View or SCADA node if the `eda.dll` and `fixtools.dll` are installed.

Note: Previous versions of this document referred to the interface as the PI-EDA Interface. The interface is the same; only the reference has changed to PI IntFix.

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\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\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 PI client applications.

Note: This interface has been built against a UniInt version (4.5.0.59 and later) which now writes all its messages to the local PI Message log.

Please see the document *UniInt Interface Message Logging for UniInt 4.5.0.x and later Interfaces* in the %PIHOME%\Interfaces\UniInt directory for more details on how to access these messages.

Note: OSIsoft is revising product documentation and other literature to reflect the evolution of the PI Server from a single server to a multi-server architecture. Specifically, the original historian core of the PI Server is now referred to as the PI Data Archive.

Originally, the PI Server was a single server that contained the PI Data Archive and other subsystems. To add features and improve scalability, the PI Server has evolved from a single server to multiple servers. While the PI Data Archive remains a core server of the PI Server product, the product name "PI Server" now refers to much more than the PI Data Archive. OSIsoft documentation, including this user manual, is changing to use "PI Server" in this broader sense and "PI Data Archive" to refer to the historian core.

Reference Manuals

OSIsoft

- PI Data Archive manuals
- *PI API Installation Instructions* manual (%PIHOME%\bin\API_install.doc)
- *PI Universal Interface (UniInt) User Guide* (%PIHOME%\Interfaces\UniInt\UniInt Interface User Manual.pdf)
- *UniInt Interface Message Logging for UniInt 4.5.0.x and later Interfaces* (%PIHOME%\Interfaces\UniInt\UniInt Interface Message Logging.pdf)
- *PI Interface Configuration Utility* User Guide (%PIHOME%\ICU\PI Interface Configuration Utility.pdf)

Intellution

- *Intellution Electronic Books*

Supported Operating Systems

Platforms		32-bit application	64-bit application
Windows 2003 Server	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows Vista	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 2008	32-bit OS	Yes	No
Windows 2008 R2	64-bit OS	Yes (Emulation Mode)	No
Windows 7	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 8 and 8.1	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 2012 Server	64-bit OS	Yes (Emulation Mode)	No

The Interface is designed to run on the above-mentioned Microsoft Windows operating systems. Because it is dependent on vendor software, newer platforms may not yet be supported.

Certain older versions of Intellution FIX/iFIX have platform restrictions that may prevent it from running on all OS listed here. Please refer to the Intellution release notes for your version.

Please contact OSIsoft Technical Support for more information.

Security Note: We recommend installing all available updates from Windows Update service. We recommend the newest versions of Windows for latest security features.

Supported Features

Feature	Support
Interface Part Number	PI-IN-INT-FIXD-NTI
Auto Creates PI Points	APS Connector
Point Builder Utility	Yes
ICU Control	Yes
PI Point Types	Float / Integer / Digital / String
* Sub-second Timestamps	Yes
* Sub-second Scan Classes	Yes
* Automatically Incorporates PI Point Attribute Changes	Yes
Exception Reporting	Yes
* Inputs to PI Data Archive	Scan-based / Unsolicited / Event Tags
Outputs to data source	Yes
Supports Questionable Bit	No
Supports Multi-character PointSource	Yes
* Maximum Point Count	Yes
Uses PI SDK	No
PINet String Support	No
* Source of Timestamps	PI Data Archive or Interface node
History Recovery	No
* UniInt-based * Disconnected Startup * SetDeviceStatus	Yes Yes Yes
* Failover	Microsoft Cluster Failover UniInt Failover Phase 1 UniInt Failover Phase 2 (Cold, Warm and Hot) Intellution SCADA Node Redundancy
* Vendor Software Required on Interface Node / PINet Node	Yes
* Vendor Software Required on Data Source Device	Yes
Vendor Hardware Required	No
* Additional PI Software Included with Interface	Yes
Device Point Types	Analog, Digital, and String
Serial-Based Interface	No

** See paragraphs below for further explanation.*

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.

Sub-second Timestamps and Scan Classes

Data will receive sub-second timestamps only if that point belongs to a scan class configured for sub-second scanning.

Automatically Incorporates PI Point Attribute Changes

The PI Point Database is checked every 2 minutes for points that are added, edited, and deleted. If point updates are detected, the points are loaded (or reloaded) by the interface as appropriate. The 2-minute update interval can be adjusted with the **`/updateinterval`** command-line parameter discussed in the UniInt Interface User Manual. The interface will only process 25 point updates at a time. If more than 25 points are added, edited, or deleted at one time, the interface will process the first 25 points, wait 30 seconds (or by the time specified by the **`/updateinterval`** parameter, whichever is shorter), process the next 25 points, and so on. After all points have been processed, the interface will resume checking for updates every 2 minutes (or by the time specified by the **`/updateinterval`** parameter). The interface will write the digital state SCAN OFF to any points that are removed from the interface while it is running.

Inputs to PI Data Archive

Data updates are either scan or event triggered. Scan based updates are collected at a frequency specified in the interface startup file. Event based updates mean an update is requested when the specified source tag receives an update. Input points can receive either scan or event based updates. Output points can only be configured for event based updates.

Alarm and event data is collected unsolicited. This data is event driven on the SCADA side and exposed to the interface as unsolicited.

Maximum Point Count

The Intellution EDA library has a tag count limit of 32,768 tags per group. Each interface scan class assigns its point list to a single EDA group. The interface therefore limits the number of points assigned to a given scan class to be less than 32,768 points.

If users need more than 32,768 points at a specific scan rate, multiple scan classes must be defined. The user will then need to distribute points among the scan classes to keep to total number of points per scan class less to than the 32,768 point count limit.

Source of Timestamps

Data is time stamped by the interface as it is received from the local Intellution node. The default behavior is that the PI Data Archive system time is used for data timestamps. Users also have the option of using the local Intellution node system time. This is configured through the interface startup file.

UniInt-based

UniInt stands for Universal Interface. UniInt is not a separate product or file; it is an OSIsoft-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 OSIsoft'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 *Universal Interface (UniInt) User Guide* is a supplement to this manual.

Disconnected Start-Up

The PI IntFix interface is built with a version of UniInt that supports disconnected start-up. Disconnected start-up is the ability to start the interface without a connection to the PI Data Archive. This functionality is enabled by adding **`/cachemode`** to the list of start-up parameters or by enabling disconnected startup using the ICU. Refer to the *PI Universal Interface (UniInt) User Guide* for more details on UniInt Disconnect startup.

SetDeviceStatus

A device status point is a type of interface Heath point. Specifically, it is a PI Data Archive point that is updated by the interface to indicate the current interface working state. For example, if a device status point exists, the interface will send an update when it establishes or loses communication with Intellution. In this way, users can monitor the device status point to track the health of the interface without referring to log files.

A device status point must be a string point and the first characters in its ExDesc attribute must be [UI_DEVSTAT]. Refer to the *UniInt Interface User Manual* for more information on configuring interface Health points.

The following events can be written to the device status point:

- “1 | Starting” – UniInt writes this string to the Device Status point when the interface starts. The snapshot for the Device Status point will contain this value until either communication is established with Intellution on the local node or the interface shuts down.
- Digital state Good – the interface writes this event to the Device Status point when it establishes communication with Intellution on the local node.
- If the interface loses communication with the Intellution on the local node, the interface writes one of the following strings to the Device Status point:
    - "3 | 1 device(s) in error | Local Intellution stopped; interface shutting down."
    - "3 | 1 device(s) in error | Local Intellution stopped; interface will continue."
- If the interface is unable to collect alarm & event data it will write one of the following updates to the Device Status point;
    - “3 | 1 device(s) in error | Unable to collect alarm & event data.”
    - “3 | 1 device(s) in error | Service library not loaded.”
- “4 | Intf Shutdown” – UniInt writes this string to the Device Status point when the interface stops.

Failover

- **Microsoft Cluster Failover Support**
  The interface supports failover through Microsoft cluster services. As with UniInt failover support, this is also a no data loss solution for bi-directional data transfer.

The significant difference is this solution requires Microsoft Cluster. See Appendix D: Cluster Failover for a complete discussion on how this works.

- **Intellution SCADA Node Redundancy**
  SCADA-node redundancy can be enabled through configuration of Intellution View nodes. In this configuration, the interface runs on a View node that connects to redundant SCADA nodes. See Appendix E: FIX Redundancy and the PI IntFix Interface for a complete discussion.

- **UniInt Failover Support**

  UniInt Phase 1 Failover provides support for a hot failover configuration which results in a *no data loss* solution for bi-directional data transfer between the PI Data Archive and the data source given a single point of failure in the system architecture. 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 1 Failover requires that the interface support output points to the Foreign System. 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.

  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 PI Data Archive 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.

Vendor Software Required

The interface can run on either an Intellution View or SCADA node if the `eda.dll` and `fixtools.dll` are installed.

It is compatible with FIX 6.15 and greater, and iFIX 2.1 and greater. The following table lists those that have been tested internally:

** Intellution Software Compatibility Testing
FIX 6.15
FIX 7.0
iFIX 2.1
iFIX 2.21
iFIX 2.6
iFIX 3.0
iFIX 3.5
iFIX 4.0
iFIX 4.5
iFIX 5.0
iFIX 5.1
iFIX 5.5
** OSIsoft will continue to test new releases of iFIX as they become available.

An issue has been discovered that prevents the PI IntFix interface from collecting data when run as a Windows service on Vista, Windows 7 and Windows Server 2008. This only occurs if the iFIX server is run interactively and not as a service. This is caused by a change in session partitioning of services and applications on the above platforms, and is described in a Microsoft white paper: http://msdn.microsoft.com/en-us/windows/hardware/gg463353.aspx.

The solution is to run the iFIX server as a service, which has been supported since iFIX 4.0. Note however that support for running as a service for Vista started with iFIX 4.5 and for Windows 7/Windows Server 2008 with iFIX 5.1 SP1A. Therefore, this issue and solution only applies to the above iFIX versions on the corresponding platforms. For supported operating systems for iFIX, please refer to the following GE KB article: http://support.ge-ip.com/support/index?page=kbchannel&id=S:KB2651.

Additional PI Software

This interface comes with the FixToPI Configuration Transfer Utility for extracting the FIX database in a format ready for exporting to PI. See Appendix C: FIXtoPI Configuration Transfer Utility for a complete discussion on this topic

The API Online (APIOnline.exe) program is also distributed along with a sample configuration file (apionline.bat_new). API Online is required for MS Cluster failover support. See Appendix D: Cluster Failover for a complete discussion on this topic.

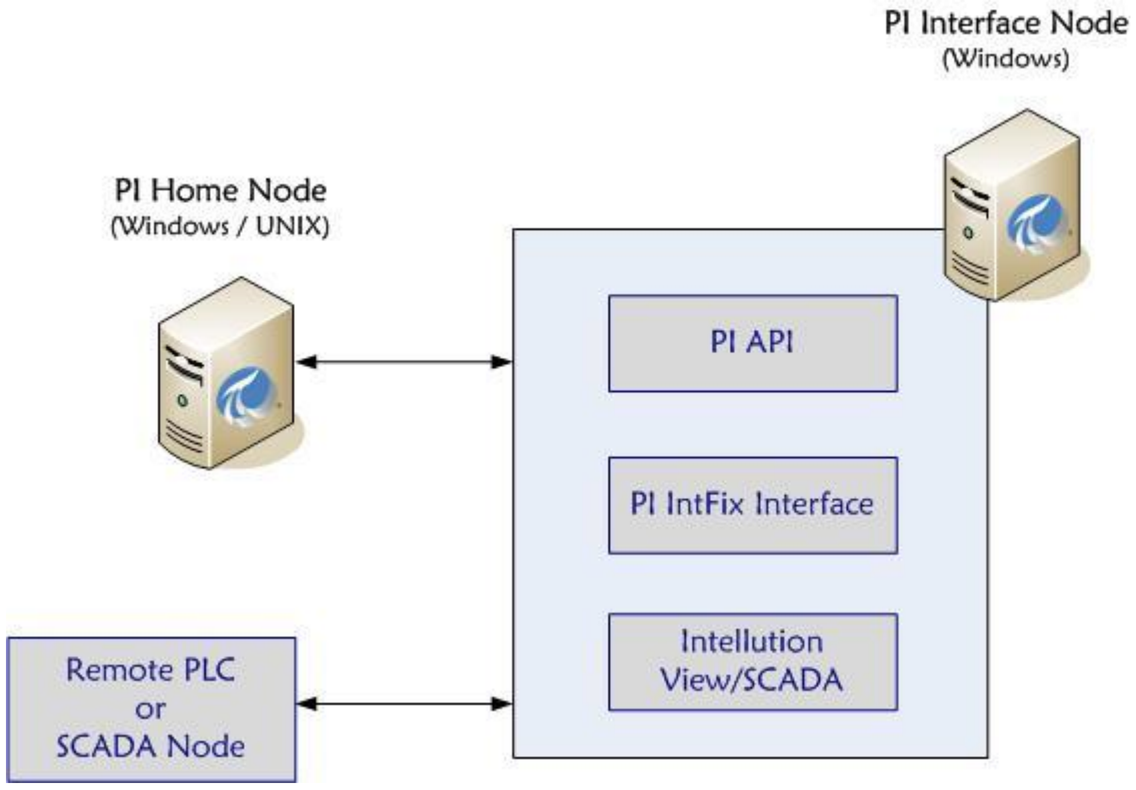
The interface installation kit also includes the OSI_iFIXmonitor program. This program serves several purposes related to coordinating the execution of PI IntFix interface instances (and PI AutoPointSync) with iFIX. Due to the design of the Intellution EDA library, a client program (like this interface) can prevent iFIX from starting. By using OSI_iFIXmonitor in conjunction with appropriate interface options, the interface can operate in a way that will not interfere with iFIX startup. See section OSI_iFixMonitor for a complete discussion on this topic.

Device Point Types

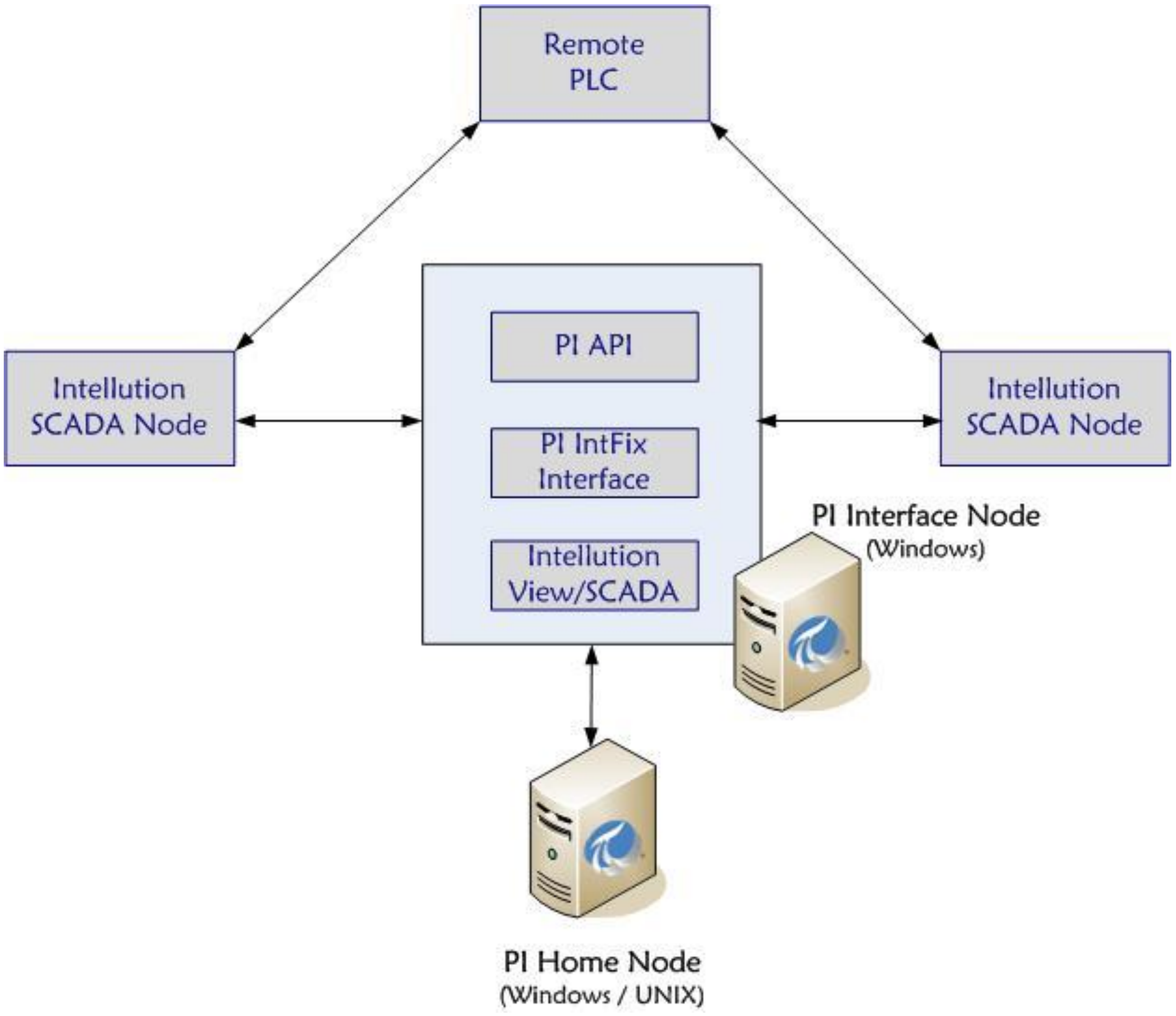
The interface can read analog, digital and string data types. Each Intellution tag has a block type which contains several fields. Each field name begins with a prefix that represents the type of data for the field. For example, Analog Input block types have a F_CV field that contains the current value. The "F_" prefix indicates that this field contains a floating point number. A complete listing of the different fields associated with each block can be found in the Intellution Database Manger Online Help file, Block Field Reference section.

Diagram of Hardware Connection

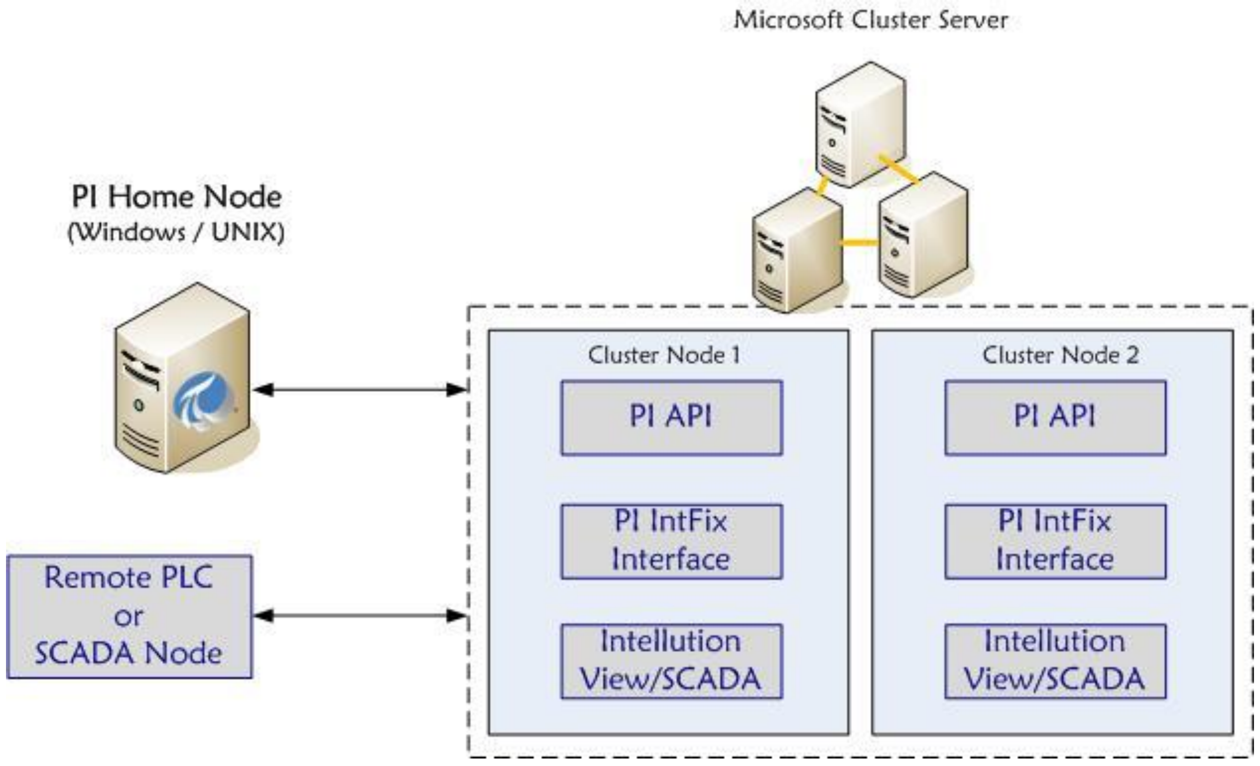
PI API Node



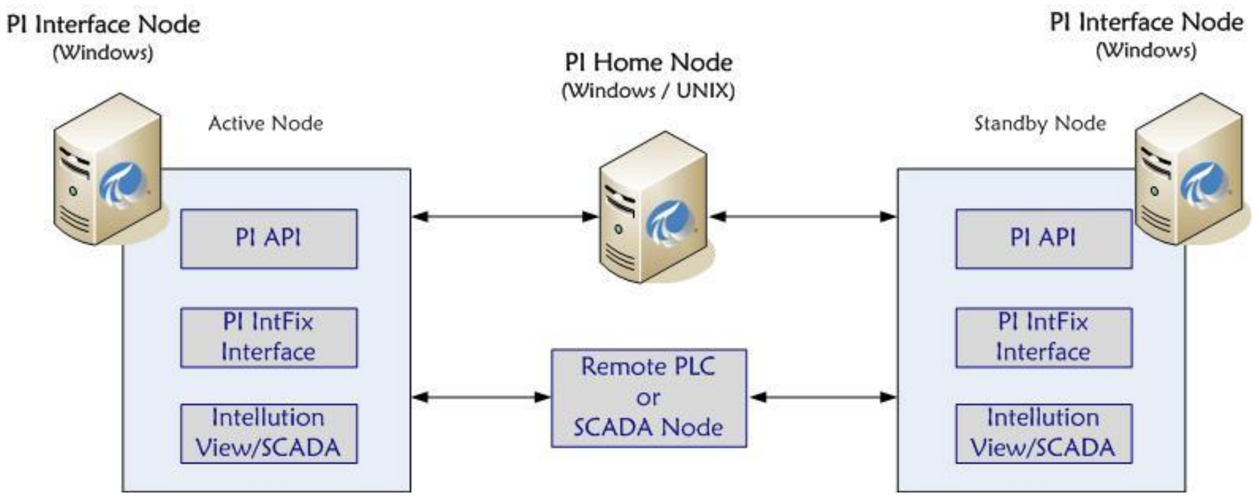
Intellution SCADA Redundancy



Microsoft Cluster Interface Failover



UniInt Interface Failover



Chapter 2. Principles of Operation

The interface uses Intellution's EDA (Easy Data Access) library to acquire data. The EDA library is common to both FIX and iFIX making this interface compatible with both platforms. The interface must run on a SCADA or View node where the eda.dll and fixtools.dll are present.

The PI IntFix interface establishes the initial connection to the PI Data Archive and reconnects in the event that the connection is lost. If the interface is started while the PI Data Archive is down, the interface will periodically try to establish a connection until successful.

When the Interface starts, the interface searches the PI Point Database for points that belong to the Interface and a point list is created for the interface.

After startup is complete, the Interface enters the processing loop, which includes:

- Servicing scheduled input points. Each Scan Class is processed in turn.
- Servicing output points as events arrive.
- Servicing triggered input points as events arrive.
- The PI Point Database is checked every 2 minutes for points that are added, edited, and deleted. If point updates are detected, the points are loaded (or reloaded) by the Interface as appropriate. The 2-minute update interval can be adjusted with the **`/updateinterval`** command-line parameter discussed in the UniInt Interface User Manual. The Interface will only process 25 point updates at a time. If more than 25 points are added, edited, or deleted at one time, the Interface will process the first 25 points, wait 30 seconds (or by the time specified by the **`/updateinterval`** parameter, whichever is lower), process the next 25 points, and so on. After all points have been processed, the Interface will resume checking for updates every 2 minutes (or by the time specified by the **`/updateinterval`** parameter). The Interface will write the digital state SCAN OFF to any points that are removed from the Interface while it is running.

Interface Startup

The interface reads the PI point database using the point source (**`/ps=char`**) and instance number (**`/id=#`**) to identify the interface points. It then processes the PI Data Archive point definition to identify which Intellution point it references using the "node-tag-field" (NTF) identifier. The node references the Intellution node name which reads data for the specified tag. Tag is the name of a block within the specified node, and field identifies a specific data value (and its type) in the block. The interface then groups these points according to scan class, with one EDA group defined for each scan class. In addition, if output points are defined, they will be placed in a separate group.

Data Updates

Data updates are either scan or event triggered. Scan based updates are collected at a frequency specified in the interface startup file. Event based updates mean an update is requested when the specified source tag receives an update. Input points can receive either scan or event based updates. Output points can only be configured for event based updates. Alarm and Event data is collected by the interface in an unsolicited manner. This data is event driven on the SCADA side and exposed to the interface unsolicited.

To optimize performance, points belonging to a particular node should be grouped into the same scan classes for more efficient polling. By keeping all points for individual nodes within the same group, EDA does not have to poll multiple nodes in order to read values for a single scan. Note that event-triggered points take much longer to process since a separate group is defined for each event point, which is less efficient than scan-based updates.

Alarm/Event Message Data Collection

The interface can also collect alarm/event message data from the Intellution WUSERQ application. In order to enable this functionality, either `WUSERQ1.exe` or `WUSERQ2.exe` must be added as a start-up task for the Intellution View or SCADA node. These tasks are responsible for making alarm/event message data available to clients. OSIsoft recommends running a separate copy of the interface specifically for alarm/event message data collection to maximize performance.

Note the latest release of the interface has been updated to prevent alarm data loss for out of order data. Alarm data is reported with a timestamp generated by Intellution. It’s possible for these Alarm events to arrive at the interface out of order. The PI Data Archive may reject out of order data if a value already exists at that time. The interface has been updated to send out of order data in ‘append’ mode to prevent alarm event data loss.

Point-by-point Alarm/Event Message Data

If enabled, the interface uses scan class one to group all PI Data Archive points that will receive alarm/event message data on a point-by-point basis.

Note: It is critical that when alarm/event message data collection is enabled, only points intended for collection of alarm/event data belong to scan class one.

In this configuration, the interface receives an alarm/event message string. This alarm string also contains the name of the Intellution source tag name. If a PI Data Archive point that belongs to scan class 1 is configured for this Intellution tag, the interface attempts to extract the data value from the string message and send it to this PI Data Archive point. The interface startup file contains parameters for defining the string position for the data within the alarm/event message (see section Startup Command File for details).

Alarm/Event Message Data to a Single PI String Point

The interface can also be configured to send all alarm/event messages to a single PI string point. The entire alarm/event message string for all events pulled from the WUSERQ are sent to a PI string point.

Warning: New configuration requirement for all alarm/event message point. Previously users specified this point name in the interface startup file. For UniInt Phase 2 failover support, this point now requires new configurations as specified in the table below. If users do not update their configuration for this point it will be rejected on startup and will not receive data.

This point must have the following configuration:

Point Attribute	Configuration
Location1	Interface ID as specified in PI-EDA.bat
Location2	2
Location4	1
PointSource	Point source as specified in PI-EDA.bat.
PointType	String

Preventing Interference with iFIX Startup

Any program that uses the Intellution EDA library for iFIX, like this interface, can prevent iFIX itself from starting. This hazard is inherent in the implementation of the EDA library and the interface provides options to co-ordinate with iFIX to avoid the problem.

Note: The default options for the interface do not co-ordinate with iFIX and will prevent iFIX from starting if the interface is running when iFIX is launched.

The fundamental issue is that, once a program loads the EDA library and calls it, the EDA library acquires resources whose existence will prevent iFIX from starting if iFIX is not already running. Once acquired, the resources held by the EDA library cannot be released programmatically and are only released when the program terminates. If iFIX stops while any programs that have called the EDA library are running, iFIX will refuse to restart until these EDA client programs terminate and consequently release the EDA library resources. The implication is that the EDA library expects EDA client programs to start after iFIX starts and stop when iFIX indicates that it is stopping. This is contrary to the typical installation of most PI interfaces, which are configured as services that start automatically with Windows and run continuously.

To avoid the situations that prevent iFIX from starting, the PI IntFix interface must 1) wait until iFIX is known to be running before the EDA library is loaded or called, and 2) terminate if it detects that iFIX has shut down after the EDA library has been called.

In order for the PI IntFix interface to start before iFIX and not prevent iFIX from subsequently starting, the interface must verify that iFIX is running before the EDA library is loaded or called. Therefore, the EDA library cannot be called to determine whether iFIX is running, and the interface must use some other method. When the PI IntFix interface is configured to wait until iFIX is running before dynamically loading the EDA library (**`/DelayLoadEDA`** parameter), the interface looks for a running copy of the OSI_iFIXmonitor program (which is included in the interface installation kit) as an indication that iFIX is running. To accurately reflect the running or stopped state of iFIX, OSI_iFIXmonitor must be configured in iFIX as a task that iFIX starts and stops. Thus, OSI_iFIXmonitor starts after iFIX is running and terminates prior to iFIX itself stopping. (Instructons for configuring OSI_iFIXmonitor are in section Configuring OSI_iFIXmonitor Program. Additional information on the OSI_iFIXmonitor program is in Appendix F: OSI_iFIXmonitor Program)

As noted earlier, once a program loads and calls the EDA library, the EDA library acquires and holds resources for the life of the process. If iFIX stops, it will not restart until all EDA client programs terminate and consequently release the iFIX resources they hold. The PI IntFix interface provides an option to terminate when it detects that iFIX has stopped (**`/StopWithFIX`** parameter). With this option, the interface terminates when iFIX stops, which releases the resources held by the EDA library so that the interface does not prevent iFIX from restarting. However, after the interface stops, data collection will not resume when iFIX restarts. The **`/StopWithFIX`** parameter is primarily intended for use with copies of the PI IntFix interface that are not configured as Windows services.

For a typical PI interface installation, the interface is configured as a Windows service so that it runs continuously and collects data whenever its data source is active. Since the constraints of the EDA library require the PI IntFix interface to terminate when iFIX stops, an external agent is needed to restart the interface service. To meet these needs, OSI_iFIXmonitor provides options to start, stop, and optionally restart the interface service in coordination with iFIX starting and stopping. When OSI_iFIXmonitor manages a PI IntFix interface service, the **`/StopWithFIX`** parameter is not necessary.

Data Redundancy

There are two distinct types of data redundancy: interface level failover and SCADA node level failover. The Intellution View/SCADA node environment supports running in a failover configuration. SCADA node failover provides the interface with two paths to PLC process data. Interface level failover ensures that PI IntFix is running in order to collect this PLC process data. PI IntFix interface failover is implemented in one of two ways: through OSIsoft UniInt Phase 1 or Phase 2 failover or by running the interface in a Microsoft Cluster environment.

Intellution SCADA Node Redundancy

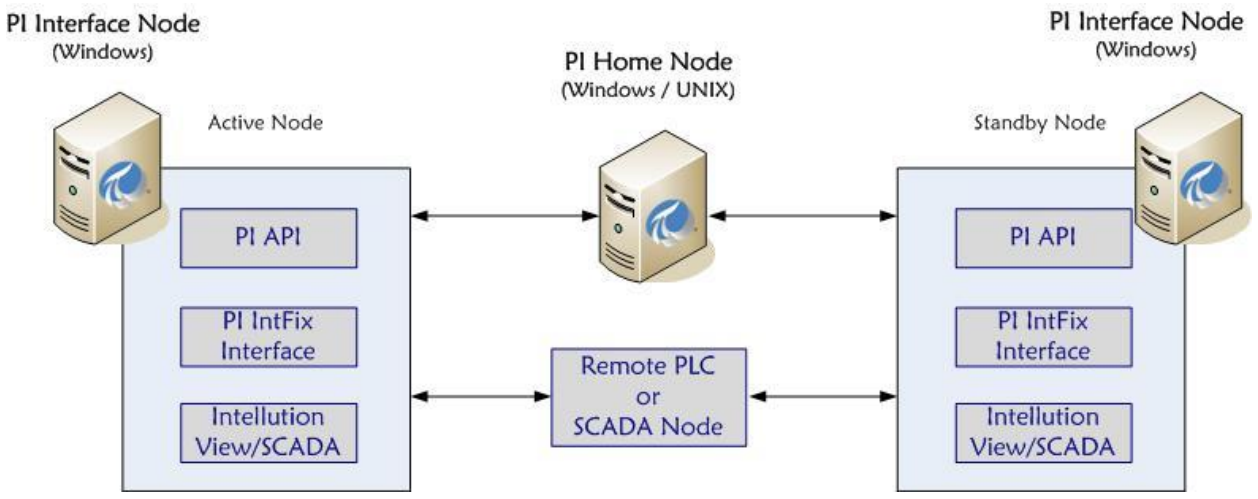
SCADA node redundancy provides the interface with two paths to PLC data. In this configuration, the interface runs on a View node which is connected to redundant SCADA failover nodes. Both FIX32 and iFIX support SCADA node failover (starting from FIX32 version 6.15 and iFIX Dynamics version 2.0). A View node can look at a pair of SCADA nodes that have identical databases (and thus are connected to the same PLC) and obtain data from the active node. More information on failover can be found in Intellution's documentation for FIX32 or iFIX. Although FIX32 allows a backup SCADA configuration that involves two SCADA servers without the use of a View node, the interface does not support this configuration. A complete discussion of SCADA-node failover, including configuration procedures, can be found in Appendix E: FIX Redundancy and the PI IntFix Interface.

Microsoft Cluster Failover Support

The interface supports hot failover when running in a Microsoft Cluster environment. Hot failover is a no data loss solution. A cluster is composed of two or more member nodes. Each member node of the cluster has a copy of the interface installed and running, with only one node sending data to PI at any given time. A complete discussion of cluster failover operation and configuration can be found in Appendix D: Cluster Failover.

UniInt Failover

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



UniInt Phase 1 Failover

UniInt Phase 1 failover is a hot failover solution. When enabled both copies of the interface collect data in parallel with only the active interface sending data to PI. Each interface instance will update an output tag on the Intellution system which acts as a heartbeat indicator. By monitoring each other’s heartbeat point the interfaces are able to determine if it should be active or not. The biggest difference between UniInt Phase 1 failover and UniInt Phase 2 failover is UniInt Phase 2 failover does not require output points on the Intellution system. Instead UniInt Phase 2 failover uses a shared file for communicating heartbeat and status information to determine interface health. Also UniInt Phase 2 failover supports hot and warm failover. Warm failover means the standby interface does not collect real-time input data which reduces loading on the SCADA system. However warm failover is not a no-data loss solution.

UniInt Phase 2 Failover

UniInt Phase 2 failover supports both warm and hot failover modes. When enabled both copies of the interface will communicate with each other through a shared file. This shared file can reside on one of the interface nodes or on a separate machine.

Hot failover means that both interfaces actively collect data with only the active interface sending its data to PI. Hot failover is a no-data loss solution.

Warm failover means that both interfaces load their point list. The standby interface will not update its point list for real-time inputs. It will however collect alarm and event data but not send it to PI unless it becomes the active interface. Warm failover should be used to minimize the impact of interface failover on the SCADA node and network loading. However warm failover is not a no-data loss solution.

Chapter 3. Installation Checklist

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

This checklist summarizes the steps for installing this Interface. You need not perform a given task if you have already done so as part of the installation of another interface. For example, you only have to configure one instance of Buffering for every interface 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 PI SMT to configure the PI Data Archive. You need not run PI SMT on the same computer on which you run this Interface.
2. If you are running the Interface on an interface node, edit the PI Data Archive's Trust Table to allow the Interface to read attributes and point data. If a buffering application is not running on the interface node, the PI trust must allow the interface to write data.
3. Run the installation kit for the PI 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 PI Data Archive to reject the data that this Interface writes.
6. If the interface is installed on an Intellution iFIX node, configure iFIX to start OSI_iFIXmonitor.
7. 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=`*`#`*)
   PI Data Archive (`/Host=`*`host:port`*)
   Scan Class(`/F=`*`##:##:##,offset`*)
8. If you will use digital points, define the appropriate digital state sets.

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

   `Location1` specifies the interface instance as specified in the startup file (**`/id=#`**).
   `Location2` specifies the I/O type (Input=0, Output=1, All Alarm & Event Tag=2).
   `Location3` is not used.
   `Location4` specifies the scan class. Event-based and output points should have `Location4=0`. Alarm/event-message points must have `Location4=1`.
   `Location5` is not used.
   `ExDesc` specifies a 'trigger tag'. It takes the format event=trigger tag.
   `InstrumentTag` specifies the NTF address (Node-Tag-Field). For example: LocalNode,TagX,F_CV.

   PtSecurity must permit read access for the PI identity, group, or user configured in the PI trust that is used by the interface.
   DataSecurity must permit read access (buffering enabled) or read/write access (unbuffered) for the PI identity, group, or user configured in the PI trust that is used by the interface.

   Security Note: When buffering is configured, the DataSecurity attribute must permit write access for the *buffering application's* PI trust or mapping. DataSecurity write permission for the interface's PI trust is required only when buffering is not configured.

10. Start the Interface interactively and confirm its successful connection to the PI Data Archive without buffering. (The DataSecurity attribute for interface points must permit write access for the interface's PI trust.)
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 PI Data Archive node. (The DataSecurity attribute for interface points must permit write access for the *buffering application's* PI trust or mapping. The interface's PI trust does not require DataSecurity write permission.)
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 PI 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 PI Data Archive node.
7. Configure the Interface Status point.

Advanced Interface Features

1. Configure the interface for Disconnected Startup. Refer to the *PI Universal Interface (UniInt) User Guide* for more details on UniInt Disconnect startup.
2. Configure UniInt failover; see the UniInt Failover Configuration chapter in this document for details related to configuring the interface for failover.

Chapter 4. Interface Installation

OSIsoft recommends that interfaces be installed on interface nodes instead of directly on the PI Data Archive node. An interface node is any node other than the PI Data Archive node where the PI Application Programming Interface (PI API) is installed (see the PI API manual). With this approach, the PI Data Archive need not compete with interfaces for the machine's resources. The primary function of the PI Data Archive 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 PI Data Archive node. In this case, the typical procedure is to install the PI Data Archive 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 PI Data Archive node. Bufserv or PIBufss can be enabled on the PI Data Archive node so that interfaces on the PI Data Archive node do not need to be started and stopped in conjunction with the PI Data Archive, but it is not standard practice to enable buffering on the PI Data Archive node. The PI Buffer Subsystem can also be installed on the PI Data Archive. See the *PI Universal Interface (UniInt) User Guide* for special procedural information.

Naming Conventions and Requirements

In the installation procedure below, it is assumed that the name of the interface executable is `PI-EDA.exe` and that the startup command file is called `PI-EDA.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, `PI-EDA1.exe` and `PI-EDA1.bat` would typically be used for interface number 1, `PI-EDA2.exe` and `PI-EDA2.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

32-bit Interfaces

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\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. OSIsoft recommends using the paths shown above as the root PIHOME directory name.

Security Note: Restrict the Windows accounts that can create or write files in the %PIHOME% folder and subfolders.

Interface Installation Directory

The interface install kit will automatically install the interface to:

```
PIHOME\Interfaces\pi-eda\
```

PIHOME is defined in the pipc.ini file.

Interface Installation Procedure

The IntFix Interface setup program uses the services of the Microsoft Windows Installer. Windows Installer is a standard part of Windows 2000 and later operating systems. When running on Windows NT 4.0 systems, the PI IntFix setup program will install the Windows Installer itself if necessary. To install, run the appropriate installation kit.

```
IntFix_#.#.#.#_.exe
```

PI Trust for Interface Authentication

A PI Interface usually runs on an interface node as a Windows service, which is a non-interactive environment. In order for an interface to authenticate itself to a PI Data Archive and obtain the access permissions for proper operation, the PI Data Archive must have a PI trust that matches the connection credentials of the interface. Determine if a suitable PI trust for the interface exists on the PI Data Archive. If a suitable PI trust does not exist, see the Security chapter for instructions on creating a new PI trust.

Installing Interface as a Windows Service

The PI IntFix Interface service can be created, preferably, with the PI Interface Configuration Utility, or can be created manually.

Security Note: We recommend running the interface service under a non-administrative account, such as a Windows built-in service virtual account, the built-in Network Service account, or a non-administrative account that you create.

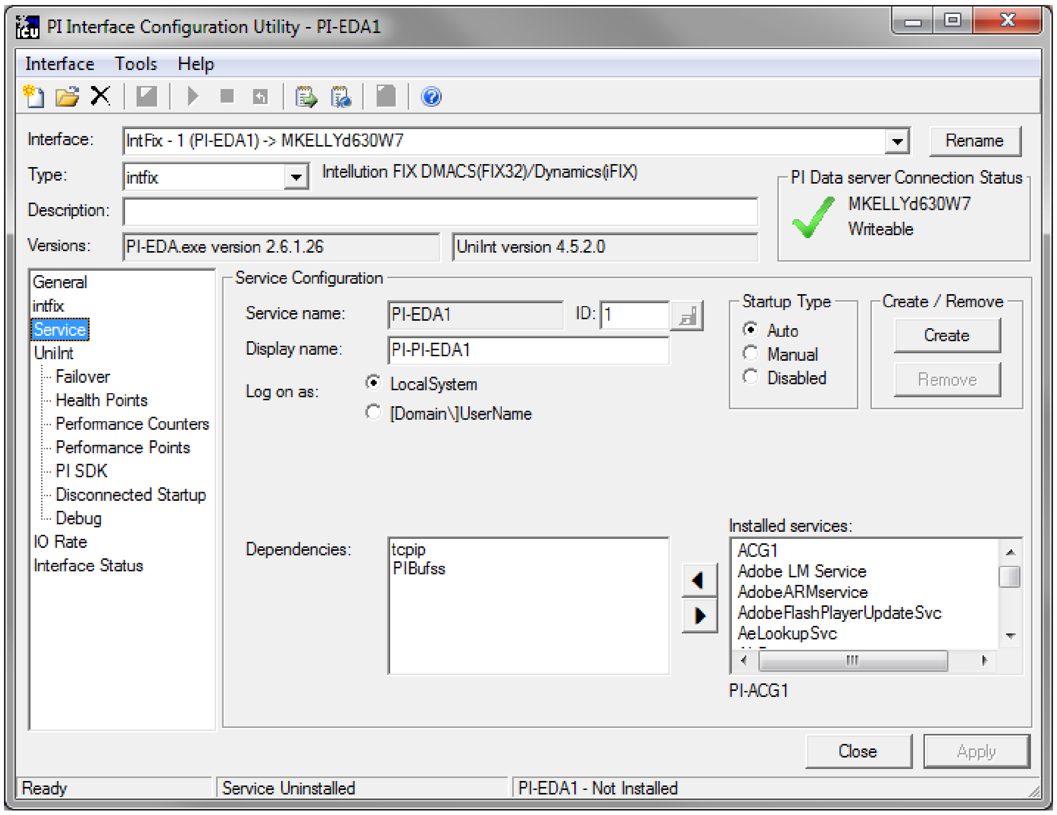
The advantage of running the interface service under an account with least privileges is improved security.

The disadvantage of running the interface service with least privileges is that, depending on the account, the interface service may not be able to create performance counters. Since UniInt health points provide essentially the same information, you may not need performance counters.

If performance counters are required, extra administrative actions are needed to create and maintain the performance counters. Since performance counters are associated with each scan class, performance counters for the interface instance must be created or deleted after adding or removing scan classes. Run the interface instance, at least for a short time, from an account that has sufficient privileges to create or delete performance counters.

Installing Interface Service with PI Interface Configuration Utility

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



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix
Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Description:
Versions: PI-EDA.exe version 2.6.1.26
UniInt version 4.5.2.0
MKELLYd630W7
Writeable

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

Service Configuration
Service name: PI-EDA1
ID: 1
Display name: PI-PI-EDA1
Log on as: [✓ LocalSystem] [Domain\]UserName
Startup Type: [✓ Auto] Manual Disabled
Dependencies: tcpip, PIBufss
Installed services: ACG1, Adobe LM Service, AdobeARMservice, AdobeFlashPlayerUpdateSvc, AeLookupSvc
PI-ACG1

Ready | Service Uninstalled | PI-EDA1 - Not 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 “PI-” prefix. Users may specify a different Display Name. OSIsoft suggests that the prefix “PI-” be appended to the beginning of the interface to indicate that the service is part of the OSIsoft suite of products.

Log on as

The *Log on as* box shows the current “Log on as” Windows 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 account for the service to use.

Security Note: For best security, we recommend running this interface service under an account with minimum privileges, such as a Windows built-in service virtual account, the built-in Network Service account, or a non administrative account that you create.

PI ICU versions earlier than 1.4.14.x cannot create a service that runs as a Windows built-in service virtual account or the built-in Network Service or Local Service accounts. After ICU creates the interface service, you can change the account with a Windows administrative tool, such as Services on the Control Panel or the sc command-line utility.

As discussed earlier, following the recommendation to run the interface service under a low-privilege account may affect performance counters.

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 PI 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*.

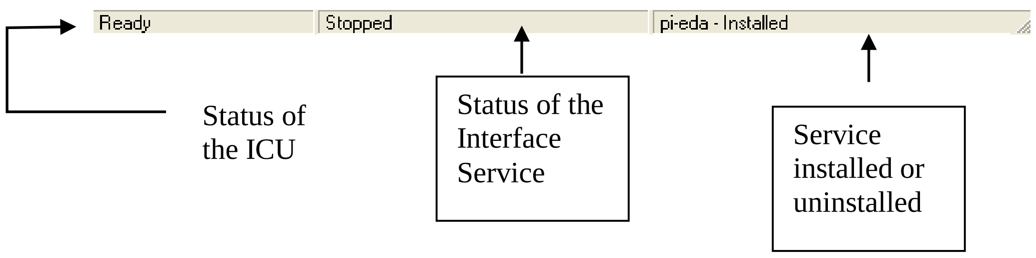
Remove

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 PI ICU dialog.


Ready
Stopped
pi-eda - 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:

```
PI-EDA.exe /help
```

Open a Windows command prompt window and change to the directory where the `PI-EDA1.exe` executable is located. Then, consult the following table to determine the appropriate service installation command.

Windows Service Installation Commands on a Interface Node or a PI Data Archive Node with Bufserv implemented	
Manual service	`PI-EDA.exe` /install /depend "tcpip bufserv"
Automatic service	`PI-EDA.exe` /install /auto /depend "tcpip bufserv"
*Automatic service with service id	`PI-EDA.exe` /serviceid X /install /auto /depend "tcpip bufserv"
Windows Service Installation Commands on a Interface Node or a PI Data Archive Node without Bufserv implemented	
Manual service	`PI-EDA.exe` /install /depend tcpip
Automatic service	`PI-EDA.exe` /install /auto /depend tcpip
*Automatic service with service id	`PI-EDA.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.

The service installation commands in this section always create an interface service that runs under the built-in Local System account. The Local System account is highly privileged and the interface does not need most of the Local System privileges to operate correctly.

Security Note: For best security, we recommend running this interface service under an account with minimum privileges, such as a Windows service virtual account, the built-in Network Service account, or a non administrative account that you create.

As discussed earlier, following the recommendation to run the interface service under a low-privilege account may affect performance counters.

The services control panel can change the account that the interface service runs under. Changing the account while the interface service is running does not take effect until the interface service is restarted.

Chapter 5. Digital States

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

Digital State Sets

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

An interface point that contains discrete data can be stored in the PI Data Archive as a digital point. A digital point associates discrete data with a digital state set, as specified by the user.

System Digital State Set

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

Chapter 6. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For example, the string *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 PI 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 PI Point Database upon startup for every PI point that is configured with a PointSource of `Boiler1`. Before an interface loads a point, the interface usually performs further checks by examining additional PI point attributes to determine whether a particular point is valid for the interface. For additional information, see the **`/ps`** parameter. If the PI API version being used is earlier than 1.6.x or the PI Data Archive version is earlier than 3.4.370.x, the PointSource 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.

Reserved Point Sources

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

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

Chapter 7. PI Point Configuration

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

If outputs to the data source device (control system) are not needed, configure the interface instance to disable outputs from PI.

Security Note: Disabling outputs from PI defends against accidental or malicious changes to the control system.

Point Attributes

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

One PI point (PI tag) must be configured for each FIX32 or iFIX field the user wishes to read and/or write data.

This document does not discuss the attributes that configure UniInt or PI Data Archive processing for a PI point. Specifically, UniInt provides exception reporting and the PI Data Archive or PIBufss provides data compression. Exception reporting and compression are very important aspects of data collection and archiving, which are not discussed in this document.

Note: See the *PI Universal Interface (UniInt) User Guide* and PI Data Archive documentation for information on other attributes that are significant to PI point data collection and archiving.

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, PI documentation uses the terms “tag” and “point” interchangeably.

Follow these rules for naming PI points:

- The name must be unique on the PI Data Archive.
- 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 PI Data Archive, this Interface supports Tag attributes 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 PI Data Archive versions.

PI API	PI Data Archive	Maximum Length
1.6.0.2 or higher	3.4.370.x or higher	1023
1.6.0.2 or higher	Below 3.4.370.x	255
Below 1.6.0.2	3.4.370.x or higher	255
Below 1.6.0.2	Below 3.4.370.x	255

If the PI Data Archive version is earlier than 3.4.370.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.

PointSource

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

PointType

Typically, device point types do not need to correspond to PI point types. For example, integer values from a device can be sent to floating point or digital PI points. Similarly, a floating-point value from the device can be sent to integer or digital PI points, although the values will be truncated.

Float16, float32, float 64, int16, int32, digital, and string point types are supported. For more information on the individual point types, see PI Data Archive manuals.

Location1

Location1 indicates to which copy of the Interface the point belongs. The value of this attribute must match the **`/id`** command-line parameter.

Location2

This parameter identifies the direction of data flow for the point or that the point is an input point that should be used for storing all alarm and event data.

Inputs

`Location2 = 0`

Defines a point as an input point (data goes from Intellution to the PI Data Archive).

Input points are updated according to their assigned scan frequency or their update is event-triggered. See the Location4 and ExDesc attribute descriptions for details about how to configure the update method for input points.

Outputs

`Location2 = 1`

Defines a point as an output point (data goes from PI to Intellution).

Output point updates are event-triggered. See the Location4 attribute description and the section Output Points below for additional configuration information.

Alarm & Event Data

`Location2 = 2`

All Alarm & Event String Point

Previous versions of this interface supported this functionality through a command line parameter. When enabled the interface would write all alarm & event data to the specified PI Data Archive string point. However in order to support UniInt Phase 2 failover the interface now requires that all points are configured specifically for the interface and not passed as command line parameters.

The following configurations are required to enable all alarm & event data to be stored in a single PI Data Archive string point:

Point Attribute	Configuration
Location1	Interface ID as specified in PI-EDA.bat
Location2	2
Location4	1
PointSource	Point source as specified in PI-EDA.bat.
PointType	String

If a point is configured with **`Location2=2`** the interface will check if another point has already been specified. If it has, any additional points with **`Location2=2`** will be rejected. Next the interface will verify that the point is a string type point and assigned to scan class 1 (**`Location4=1`**). If these configurations are in place the interface uses this point to record all alarm and event message strings it collects from the local node.

Note: Alarm & event message data collection must also be enabled through the ICU in order for all alarm & event string point to receive data.

Location3

Location3 is not used by this interface.

Location4

Scan-based Inputs

For interfaces that support scan-based collection of data, Location4 defines the scan class for the PI 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.

Trigger-based Inputs, Unsolicited Inputs, and Output Points

Location 4 should be set to zero for these points.

Alarm/Event Message Data Collection

If alarm/event message data collection is enabled (see the **`/em`** and **`/c`** command line parameters), scan class 1 is used for alarm/event message data collection EXCLUSIVELY. All points with **`Location4=1`** will be used for this purpose.

Performance Considerations

The absolute limit on resolution is 0.01 second. With high data resolution (fast scan rates), users should monitor CPU loading. The higher the data resolution, the more CPU time the interface will need.

To optimize performance, points belonging to a particular Intellution node should be grouped into the same scan class for more efficient polling. By keeping all points for individual nodes within the same group, EDA does not have to poll multiple nodes in order to read values for a single scan. Note that event-triggered points take much longer to process since a separate group is defined for each event point, which is less efficient than scan-based updates.

Location5

This parameter determines the interface behavior when a NULL (blank) string value is received from Intellution for the configured source point. This provides users with an option for client viewing of string or digital data. This attribute provides users the option of having a 'No Data' value written in place of the NULL.

`Location5=0`

Send NULL string value when a NULL string value is received as an update.

`Location5=1`

Write 'No Data' system digital state when NULL string value is received as an update.

InstrumentTag

InstrumentTag is used to specify the "node,tag,field" (NTF) identifier. The node references the Intellution node name which reads data for the specified tag. Tag references a block within the specified node, and field identifies a specific data value in the block and field data type. The NTF identifier is used to map PI points to the corresponding Intellution point.

The following table shows the field values to obtain current values for given data types.

Intellution Data Type	**Field Value**	**Input Points: Supported PI Point Types**	**Output Points: Supported PI Point Types**
Analog or Integer	F_CV	Float, Integer or Digital	Float, Integer or Digital
Digital or Boolean	D_CV	Digital or Integer	Digital
Multi-state Digital	M_CV	Digital or Integer	Digital
String	A_CV	String	String

The interface has the ability to obtain a wide range of data for each block type. For a complete listing of the field options for each Intellution block type, see the Intellution Database Manger Online Help, Block Field Reference section.

The current value fields in Intellution digital and multi-state digital block types are named A_CV, where the "A_" prefix indicates that the field contains a string value. The PI IntFix Interface provides translation from the string value to the integer value corresponding to the string so that data from digital or multi-state digital blocks can be used with PI points whose PI point types are digital or integer. The translation is enabled be using special field names as discussed below.

The InstrumentTag attribute requires the following format:

`Node,Tag,Field`

The following table provides examples of how to configure the InstrumentTag given the Intellution node, tag name, and block type.

Intellution Node	Tag Name	Block Type	PI InstrumentTag Definition
PLANT1	FLOW_PV	Analog Input	PLANT1,FLOW_PV,F_CV
PLANT1	VALVE_PV	Digital Input	PLANT1,VALVE_PV,D_CV
PLANT2	CONTROL_SP	Multi-state Digital Input	PLANT2,CONTROL_SP,M_CV
PLANT2	COMMENT	String	PLANT2,COMMENT,A_CV

Note that the interface is not limited to the block types listed in the above table.

Digital blocks define two strings corresponding to binary values 0 and 1. If the field name in a PI Data Archive point definition has a "D_" type prefix, the PI IntFix Interface internally replaces the "D_" type prefix with "A_" to locate the actual Intellution string-valued field. (The "D_" type prefix is not known to the Intellution system.) The "D_" type prefix indicates to the interface that the value should be translated from string to integer. To define a PI Data Archive digital or integer point for a string-type field in an Intellution digital block, replace the "A_" type prefix in the field name with "D_". For example, the current value field in digital blocks is named "A_CV". In definitions for digital or integer PI Data Archive points for the current value of an Intellution digital block, use "D_CV" as the field name.

Multi-state blocks define up to eight strings corresponding to states 0 to 7. If the field name in a PI Data Archive point definition has a "M_" type prefix, the PI IntFix Interface internally replaces the "M_" type prefix with "A_" to locate the actual Intellution string-valued field. (The "M_" type prefix is not known to the Intellution system.) The "M_" type prefix indicates to the interface that the value should be translated from string to integer. To define a PI Data Archive digital or integer point for a string-type field in an Intellution multi-state digital block, replace the "A_" type prefix in the field name with "M_". For example, the current value field in multi-state digital blocks is named "A_CV". In definitions for digital or integer PI Data Archive points for the current value of an Intellution multi-state digital block, use "M_CV" as the field name.

Length

Depending on the version of the PI API and the PI Data Archive, 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 PI Data Archive versions.

PI API	PI Data Archive	Maximum Length
1.6.0.2 or higher	3.4.370.x or higher	1023
1.6.0.2 or higher	Below 3.4.370.x	32
Below 1.6.0.2	3.4.370.x or higher	32
Below 1.6.0.2	Below 3.4.370.x	32

If the PI Data Archive version is earlier than 3.4.370.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.

If the NTF definition exceeds the InstrumentTag length limit, the extended descriptor can be used for defining the node and field names. If the NTF entry in the InstrumentTag is not complete, the **`ExDesc`** attribute will be checked. If the full NTF is specified in the InstrumentTag, then the interface does not check the **`ExDesc`** attribute for additional information – the interface already has all the information required. Using the **`ExDesc`** attribute for this purpose means the **`InstrumentTag`** will contain the Intellution tag name and the field and node names are defined in the **`ExDesc`** attribute.

ExDesc

Length

Depending on the version of the PI API and the PI Data Archive, this Interface supports an `ExDesc` 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 PI Data Archive versions.

PI API	PI Data Archive	Maximum Length
1.6.0.2 or higher	3.4.370.x or higher	1023
1.6.0.2 or higher	Below 3.4.370.x	80
Below 1.6.0.2	3.4.370.x or higher	80
Below 1.6.0.2	Below 3.4.370.x	80

If the PI Data Archive version is earlier than 3.4.370.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.

Node and Field Definitions

The extended descriptor can also be used for defining the node and field names when the NTF definition exceeds the length limit for InstrumentTag (see InstrumentTag description for more details).

The following format should be used for defining the node and field names.

`NODE=node name, FIELD=field name`

All three (event=, node=, and field=) can be defined in the extended descriptor. The following example shows the syntax for a point specifying a trigger tag along with the node and field names (the order in which the parameters are defined is not important):

`EVENT=trigger tag, NODE=node id, FIELD=field id.`

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.

Trigger-based Inputs

For trigger-based input points, a separate trigger point must be configured. An input point is associated with a trigger point by entering a case-insensitive string in the extended descriptor (ExDesc) PI point attribute of the input point of the form:

```
keyword=trigger_tag_name
```

where `keyword` is replaced by "event" or "trig" and `trigger_tag_name` is replaced by the name of the trigger point. There should be no spaces in the string. UniInt automatically assumes that an input point is trigger-based instead of scan-based when the `keyword=trigger_tag_name` string is found in the extended descriptor attribute.

An input is triggered when a new value is sent to the Snapshot of the trigger point. The new value does not need to be different than the previous Snapshot value to trigger an input, but the timestamp of the new value must be greater than (more recent than) or equal to the timestamp of the previous value. This is different than the trigger mechanism for output points. For output points, the timestamp of the trigger value must be greater than (not greater than or equal to) the timestamp of the previous value.

Conditions can be placed on trigger events. Event conditions are specified in the extended descriptor as follows:

```
Event='trigger_tag_name' event_condition
```

The trigger tag name must be in single quotes. For example,

```
Event='Sinusoid' Anychange
```

will trigger on any event to the PI point sinusoid as long as the next event is different than the last event. The initial event is read from the snapshot.

The keywords in the following table can be used to specify trigger conditions.

Event Condition	Description
Anychange	Trigger on any change as long as the value of the current event is different than the value of the previous event. System digital states also trigger events. For example, an event will be triggered on a value change from 0 to "Bad Input," and an event will be triggered on a value change from "Bad Input" to 0.
Increment	Trigger on any increase in value. System digital states do not trigger events. For example, an event will be triggered on a value change from 0 to 1, but an event will not be triggered on a value change from "Pt Created" to 0. Likewise, an event will not be triggered on a value change from 0 to "Bad Input."
Decrement	Trigger on any decrease in value. System digital states do not trigger events. For example, an event will be triggered on a value change from 1 to 0, but an event will not be triggered on a value change from "Pt Created" to 0. Likewise, an event will not be triggered on a value change from 0 to "Bad Input."
Nonzero	Trigger on any non-zero value. Events are not triggered when a system digital state is written to the trigger point. For example, an event is triggered on a value change from "Pt Created" to 1, but an event is not triggered on a value change from 1 to "Bad Input."

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 log and the point is not loaded by the Interface. There is one exception to the previous statement.

If any PI 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 PI point regardless of the value of the Scan attribute. Two examples of actions that would remove a PI 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 point to be rejected by the Interface, `SCAN OFF` will be written to the PI point.

SourceTag

A SourceTag is used in conjunction with an output point. An output point has Location2 set to 1.

Shutdown

The Shutdown attribute is 1 (true) by default. The default behavior of the PI Shutdown subsystem is to write the `SHUTDOWN` digital state to all PI points when PI is started. The timestamp that is used for the `SHUTDOWN` events is retrieved from a file that is updated by the Snapshot Subsystem. The timestamp is usually updated every 15 minutes, which means that the timestamp for the `SHUTDOWN` events will be accurate to within 15 minutes in the event of a power failure. For additional information on shutdown events, refer to PI Data Archive manuals.

Note: The `SHUTDOWN` events that are written by the PI Shutdown subsystem are independent of the `SHUTDOWN` events that are written by the Interface when the **`/stopstat=Shutdown`** command-line parameter is specified.

`SHUTDOWN` events can be disabled from being written to PI points when the PI Data Archive is restarted by setting the Shutdown attribute to 0 for each point. Alternatively, the default behavior of the PI Shutdown Subsystem can be changed to write `SHUTDOWN` events only for PI points that have their Shutdown attribute set to 0. To change the default behavior, edit the `\PI\dat\Shutdown.dat` file, as discussed in PI Data Archive manuals.

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 PI Data Archive, allowing continuous data collection when the PI Data Archive is down for maintenance, upgrades, backups, and unexpected failures. That is, when the PI Data Archive is shutdown, Bufserv or PIBufss will continue to collect data for the Interface, making it undesirable to write `SHUTDOWN` events to the PI points for this Interface. Disabling Shutdown is recommended when sending data to a Highly Available PI Data Archive Collective. Refer to the Bufserv or PIBufss manuals for additional information.

DataSecurity

The PI identity in the PI trust that authenticates the interface must be granted read access by the DataSecurity attribute of every PI point that the interface services. If the interface is used *without* a buffering application, write access also must be granted. (If the interface is used with a buffering application, the buffering application requires write access but the interface does not.)

PtSecurity

The PI identity in the PI trust that authenticates the interface must be granted read access by the PtSecurity attribute of every PI point that the interface services.

Output Points

Output points control the flow of data from the PI Data Archive to any destination that is external to the PI Data Archive, such as a PLC or a third-party database. For example, to write a value to a register in a PLC, use an output point. Each interface has its own rules for determining whether a given point is an input point or an output point. There is no *de facto* PI point attribute that distinguishes a point as an input point or an output point.

Security Note: When output points are required, implement an output point whitelist, which provides a defense against accidental or malicious changes to the control system.

Outputs are triggered for UniInt-based interfaces. That is, outputs are not scheduled to occur on a periodic basis. There are two mechanisms for triggering an output.

As of UniInt 3.3.4, event conditions can be placed on triggered outputs. The conditions are specified using the same event condition keywords in the extended descriptor as described under "Trigger-Based Inputs. The only difference is that the trigger point is specified with the SourceTag attribute instead of with the "event" or "trig" keywords. Otherwise, the behavior of event conditions described in section Trigger-Based Inputs is identical for output points. For output points, event conditions are specified in the extended descriptor as follows:

```
event_condition
```

Trigger Method 1 (Recommended)

For trigger method 1, a separate trigger point must be configured. The output point must have the same point source as the interface. The trigger point can be associated with any point source, including the point source of the interface. Also, the point type of the trigger point does not need to be the same as the point type of the output point.

The output point is associated with the trigger point by setting the SourceTag attribute of the output point equal to the tag name of the trigger point. An output is triggered when a new value is sent to the Snapshot of the trigger point. The new value does not need to be different than the previous value that was sent to the Snapshot to trigger an output, but the timestamp of the new value must be more recent than the previous value. If no error is indicated, then the value that was sent to the trigger point is also written to the output point. If the output is unsuccessful, then an appropriate digital state that is indicative of the failure is usually written to the output point. If an error is not indicated, the output still may not have succeeded because the interface may not be able to tell with certainty that an output has failed.

Trigger Method 2

For trigger method 2, a separate trigger point is not configured. To trigger an output, write a new value to the Snapshot of the output point itself. The new value does not need to be different than the previous value to trigger an output, but the timestamp of the new value must be more recent than the previous value.

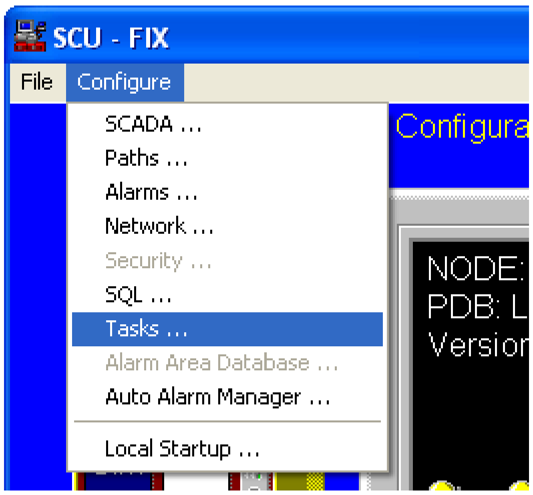
Trigger method 2 may be easier to configure than trigger method 1, but trigger method 2 has a significant disadvantage. If the output is unsuccessful, there is no point to receive a digital state that is indicative of the failure, which is very important for troubleshooting.

Chapter 8. Configuring OSI_iFIXmonitor Program

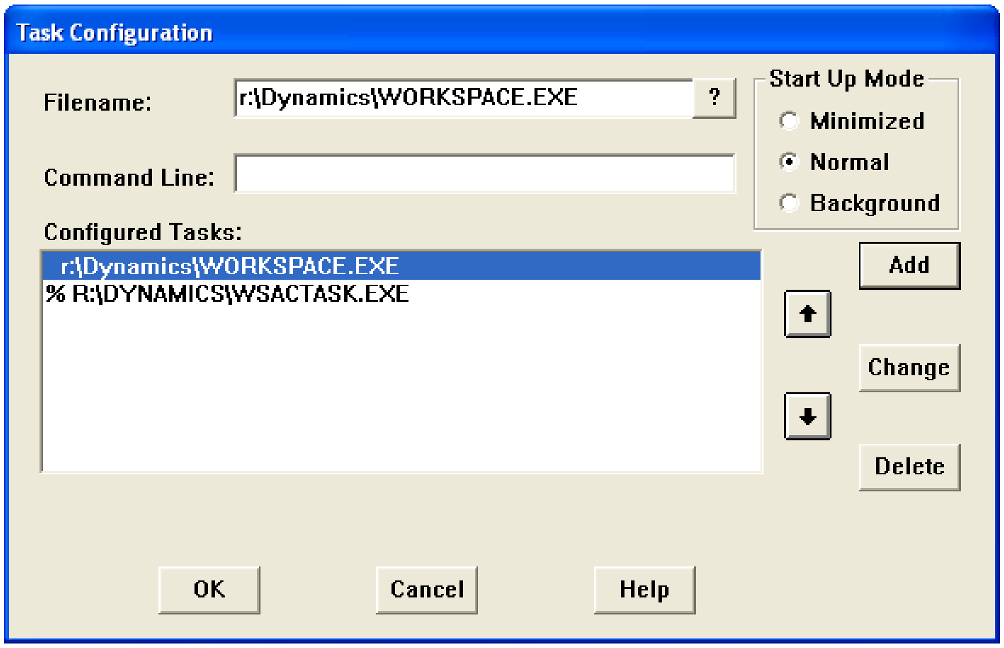
Any program that uses the Intellution EDA library for iFIX, like this interface, can prevent iFIX itself from starting. To address this problem, the OSI_iFIXmonitor program (OSI_iFIXmonitor.exe) is included in the interface installation kit and the interface provides options to coordinate with iFIX that require configuration of the OSI_iFIXmonitor program as a task that iFIX starts and stops. This section explains how to add OSI_iFIXmonitor to the iFIX task list. Additional information about the problem and OSI_iFIXmonitor program is in the Principles of Operation section and Appendix F: OSI_iFIXmonitor Program.

The interface installation kit stores a copy of the OSI_iFIXmonitor program in the same directory as the interface (see section Interface Installation Directory. To add OSI_iFIXmonitor to the iFIX task list, take the following steps.

1. Open the Intellution System Configuration Utility (SCU).
2. On the Configuration menu, select **Tasks**.



3. The **Task Configuration** dialog box opens and shows the tasks that are already configured.



Filename: r:\Dynamics\WORKSPACE.EXE

Start Up Mode: ( ) Minimized [✓ Normal] ( ) Background

Configured Tasks:

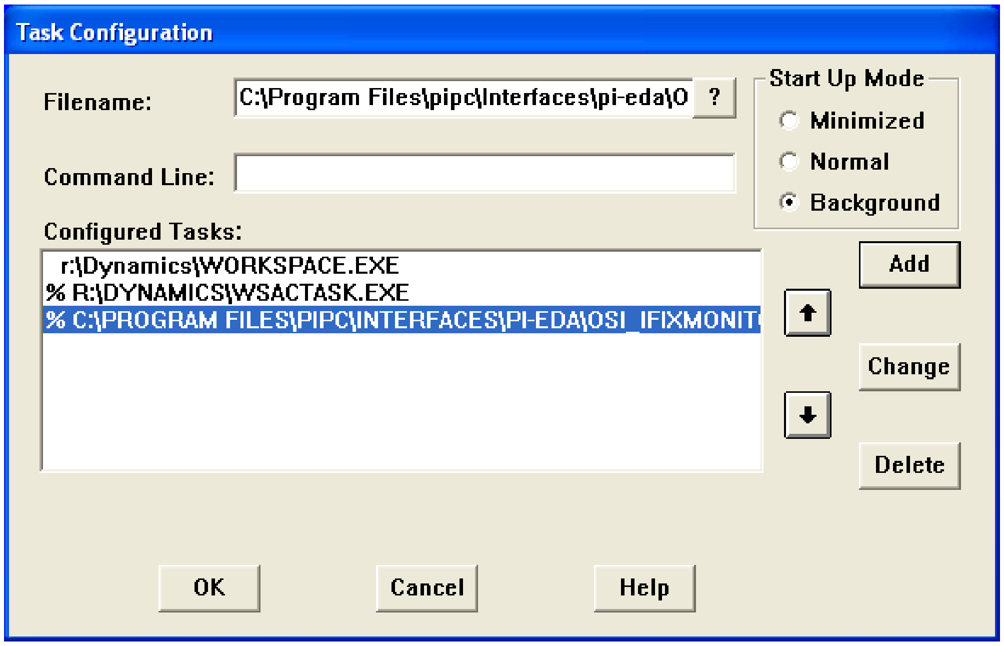
```
r:\Dynamics\WORKSPACE.EXE
% R:\DYNAMICS\WSACTASK.EXE
```

4. To add a new task, first change the **Filename** box to the full path to the OSI_iFIXmonitor program. Click ? to open a file browser, select the program file, and the path will be entered in the **Filename** box. Or, type the full path to the OSI_iFIXmonitor program in the **Filename** box.

5. The **Command Line** box contains command line parameters for the task. Typically, no command line parameters are needed for OSI_iFIXmonitor. Delete any text in the **Command Line** box. Parameters for the OSI_iFIXmonitor program are discussed in Appendix F: OSI_iFIXmonitor Program.

6. In the Start Up Mode area, select the **Background** option. If iFIX is configured to run as a service, Background must be selected. When iFIX is not configured as a service, the other options are permitted, but there is usually no reason to use them.

7. Click **Add** to create a new entry in **Configured Tasks** box.



Filename: C:\Program Files\pipc\Interfaces\pi-eda\O

Start Up Mode: ( ) Minimized ( ) Normal [✓ Background]

Configured Tasks:

```
 r:\Dynamics\WORKSPACE.EXE
% R:\DYNAMICS\WSACTASK.EXE
% C:\PROGRAM FILES\PIPC\INTERFACES\PI-EDA\OSI_IFIXMONIT
```

8. Click **OK** to close the **Task Configuration** dialog box.
9. Exit from the System Configuration Utility and save changes.
10. If iFIX is running, it must be stopped and restarted to start the OSI_iFIXmonitor task. The OSI_iFIXmonitor program writes messages to the log file when it starts or stops, which can be used to confirm that OSI_iFIXmonitor is being started and stopped by iFIX. Also, the list on the Windows Task Manager Processes tab should include `OSI_iFIXmonitor.exe` while iFIX is running.

Chapter 9. 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.

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

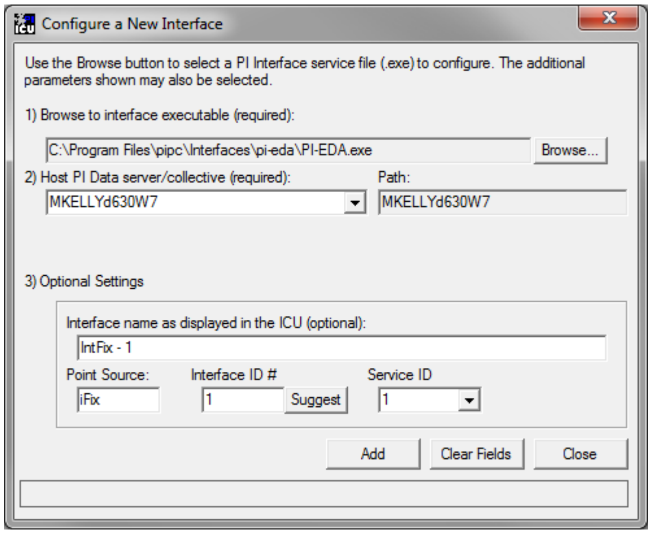
The PI Interface Configuration Utility (PI ICU) provides a tool for configuring the Interface startup command file.

Configuring the Interface with PI ICU

Note: PI ICU requires PI 3.3 or greater.

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

From the PI ICU menu, select *Interface*, then *NewWindows Interface Instance from EXE...*, and then *Browse* to the `PI-EDA.exe` executable file. Then, enter values for *Host PI Data server/collective, Point Source* and *Interface ID#*. A window such as the following opens:



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\pi-eda\PI-EDA.exe

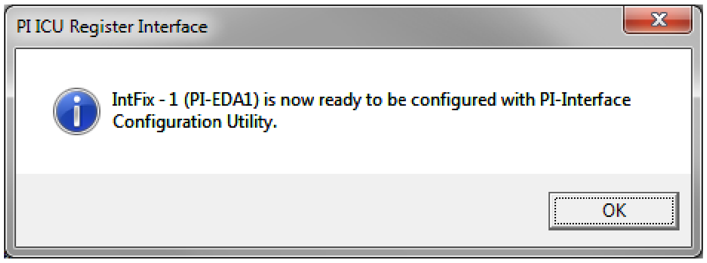
2) Host PI Data server/collective (required): MKELLYd630W7
Path: MKELLYd630W7

3) Optional Settings
Interface name as displayed in the ICU (optional): IntFix - 1
Point Source: iFix
Interface ID #: 1
Service ID: 1

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

Click on *Add*.

The following display should appear:

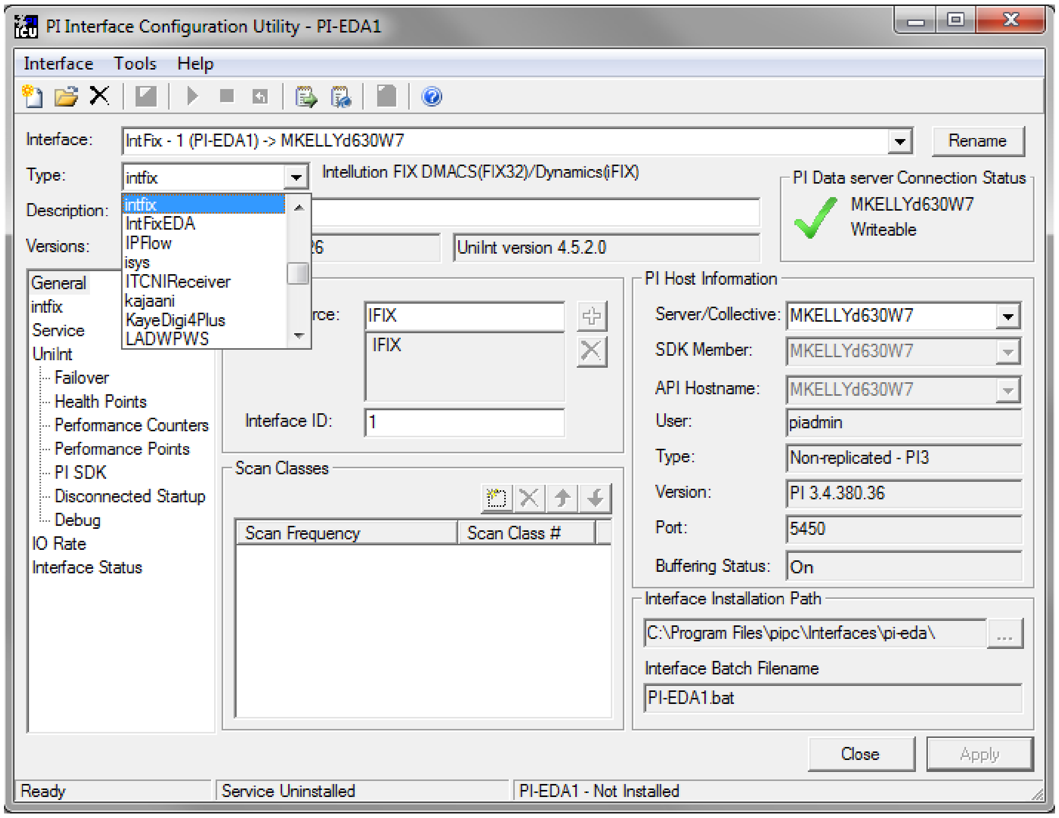

PI ICU Register Interface

IntFix - 1 (PI-EDA1) is now ready to be configured with PI-Interface Configuration Utility.

Note that in this example the Host PI Data server is MKELLYd630W7. To configure the interface to communicate with a remote PI Data server, select '*Interface => Connections…*' item from PI 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 PI ICU, near the top of the main PI ICU screen, the Interface *Type* should be `IntFix`. If not, use the drop-down box to change the Interface *Type* to be `IntFix`

Click on *Apply* to enable the PI ICU to manage this copy of the IntFix Interface.


PI Interface Configuration Utility - PI-EDA1

Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix — Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Versions: …26 — UniInt version 4.5.2.0

PI Data server Connection Status: MKELLYd630W7 Writeable

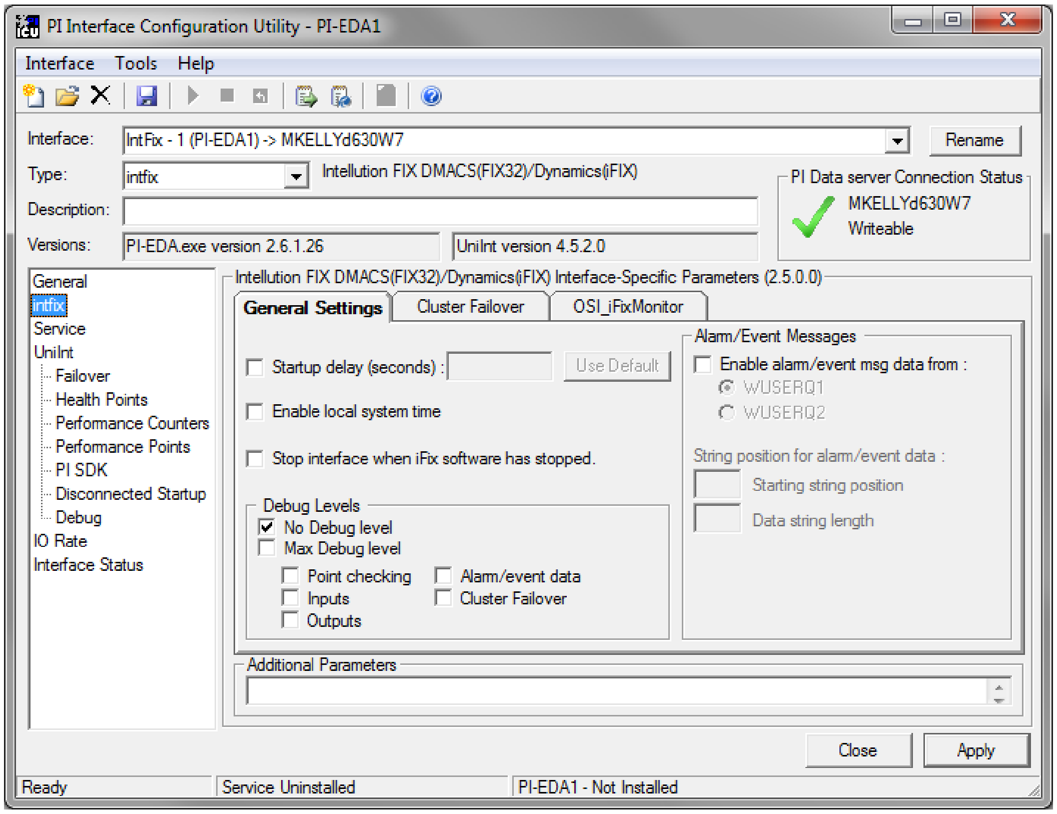
Interface ID: 1

PI Host Information
Server/Collective: MKELLYd630W7
SDK Member: MKELLYd630W7
API Hostname: MKELLYd630W7
User: piadmin
Type: Non-replicated - PI3
Version: PI 3.4.380.36
Port: 5450
Buffering Status: On

Interface Installation Path: C:\Program Files\pipc\Interfaces\pi-eda\
Interface Batch Filename: PI-EDA1.bat

Ready | Service Uninstalled | PI-EDA1 - Not Installed

The next step is to make selections in the interface-specific tab (i.e. "`IntFix`") that allow the user to enter values for the startup parameters that are particular to the PI IntFix Interface.



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix — Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Description:
Versions: PI-EDA.exe version 2.6.1.26 — UniInt version 4.5.2.0

PI Data server Connection Status: MKELLYd630W7, Writeable

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

Intellution FIX DMACS(FIX32)/Dynamics(iFIX) Interface-Specific Parameters (2.5.0.0)

General Settings | Cluster Failover | OSI_iFixMonitor

- ☐ Startup delay (seconds) : [Use Default]
- ☐ Enable local system time
- ☐ Stop interface when iFix software has stopped.

Debug Levels
- ☑ No Debug level
- ☐ Max Debug level
  - ☐ Point checking
  - ☐ Inputs
  - ☐ Outputs
  - ☐ Alarm/event data
  - ☐ Cluster Failover

Alarm/Event Messages
- ☐ Enable alarm/event msg data from :
  - ● WUSERQ1
  - ○ WUSERQ2

String position for alarm/event data :
- Starting string position
- Data string length

Additional Parameters

Close | Apply

Ready | Service Uninstalled | PI-EDA1 - Not Installed

Since the PI IntFix Interface is a UniInt-based interface, in some cases the user will need to make appropriate selections in the **UniInt** page. This page allows the user to access UniInt features through the PI ICU and to make changes to the behavior of the interface.

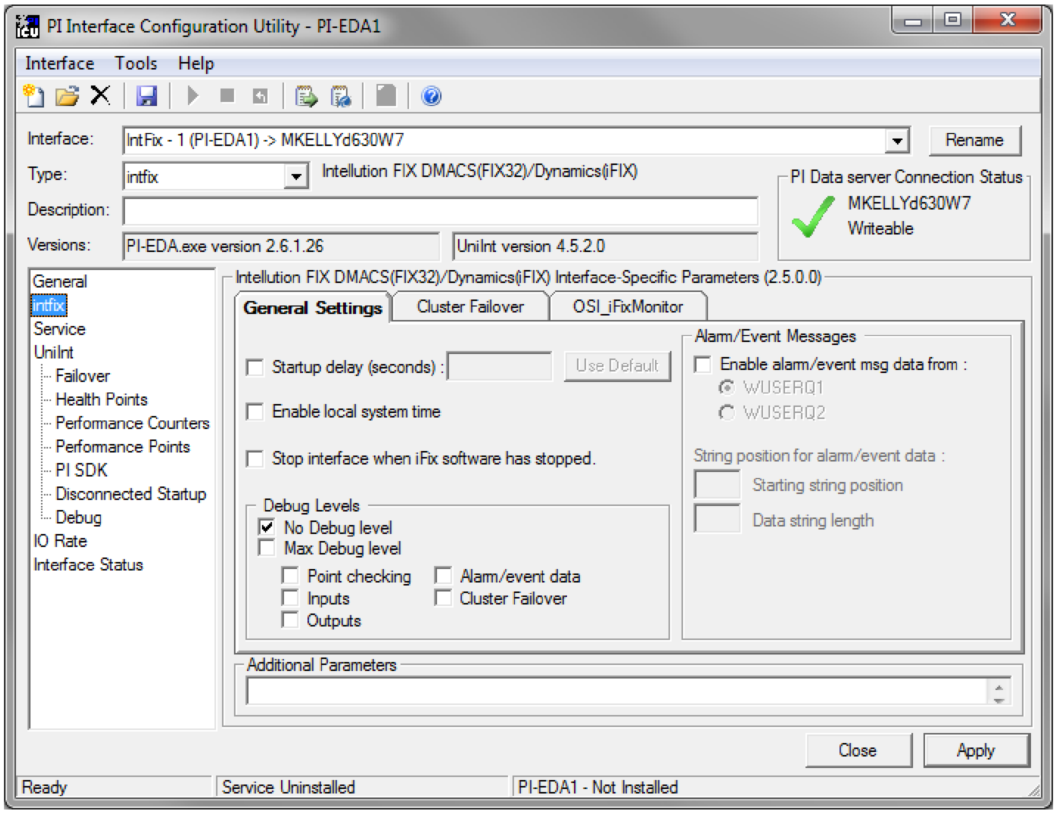
To set up the interface as a Windows Service, use the **Service** page. This page allows configuration of the interface to run as a service as well as to starting and stopping of the interface. The interface can also be run interactively from the PI ICU. To do that go to menu, select the Interface item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

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

Intfix Interface page

Since the startup file of the PI IntFix Interface is maintained automatically by the PI ICU, use the *intfix* 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 PI ICU Control and corresponding manual parameters.

Intfix



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Description:
Versions: PI-EDA.exe version 2.6.1.26 UniInt version 4.5.2.0
PI Data server Connection Status: MKELLYd630W7 Writeable

Intellution FIX DMACS(FIX32)/Dynamics(iFIX) Interface-Specific Parameters (2.5.0.0)

Startup delay (seconds) :
Enable local system time
Stop interface when iFix software has stopped.

Debug Levels
[✓] No Debug level
Max Debug level
Point checking
Alarm/event data
Inputs
Cluster Failover
Outputs

Alarm/Event Messages
Enable alarm/event msg data from :
[● WUSERQ1] WUSERQ2
String position for alarm/event data :
Starting string position
Data string length

Additional Parameters

Ready | Service Uninstalled | PI-EDA1 - Not Installed

The PI IntFix ICU Control for PI ICU has three tabs. A yellow text box indicates that an invalid value has been entered, or that a required value has not been entered.

General Settings

Startup delay (seconds)

Enabling the check box allows you to specify how many seconds the interface waits on startup before connecting to Intellution. The default is 120. The delay allows the Intellution software to fully start before trying to connect (`/W=delay`, default:120).

Enable local system time

The default behavior of the interface is to use the PI Data Archive system time for the data timestamp. Enable this check box to have the interface use the local interface node system time for timestamp source.

This option must be used with caution. If the local system time is ahead of the PI Data Archive, the data may be rejected. PI discards “future” data, which is defined as any event more than 10 minutes ahead of the PI Data Archive time. This option should only be used if there is a compelling reason to do so (`/LS`)

Stop interface when iFix software has stopped

This option will cause the interface to stop if the iFix software is stopped (`/STOPWITHIFIX`).

Debug Levels

The interface has the option of enabling debug messaging for specific operations. Selecting **Max debug level** enables messaging for all specific operations plus additional messaging. Click on the appropriate check box to enable the desired debug messages. Note that enabling **Point checking** will slow interface startup proportional to the number of points; specifically, more points means slower interface startup (`/DB=#,#`,… Range:0-6).

Alarm/Event Messages

Note: When alarm/event message data collection is enabled, all points belonging to scan class one will be used for this purpose.

It is recommended that a separate copy of the interface be run specifically for the purpose of collecting alarm/event message data.

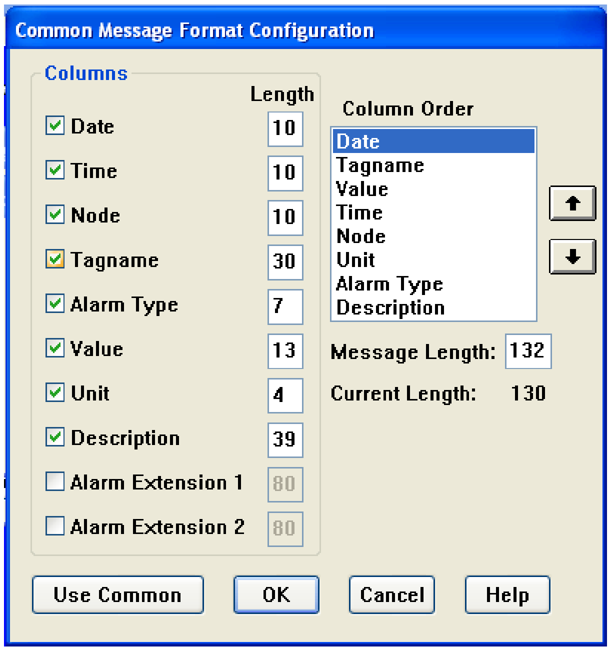
Enable alarm/event msg data from

The check box allows you to configure the interface to collect alarm/event message data. Once this check box is active, it will enable the radio buttons for specifying which WUSERQ to query (`/EM` Default=WUSERQ1).

String position for alarm/event data

In order to collect alarm/event data on a tag-by-tag basis, the string position and length of the data value within the message string must be specified.

Refer to the **Alarm Common Message Format Configuration box** accessed through the Intellution System Configuration Utility. On the **Configure** menu, click **Alarms** to open the **Alarm Configuration** dialog box. Click **Advanced** to open the **Advanced Alarm Configuration** box. Click **Common Format** to open the following dialog box:


Common Message Format Configuration

Columns

	Length
[✓] Date	10
[✓] Time	10
[✓] Node	10
[✓] Tagname	30
[✓] Alarm Type	7
[✓] Value	13
[✓] Unit	4
[✓] Description	39
[ ] Alarm Extension 1	80
[ ] Alarm Extension 2	80

Column Order
- Date
- Tagname
- Value
- Time
- Node
- Unit
- Alarm Type
- Description

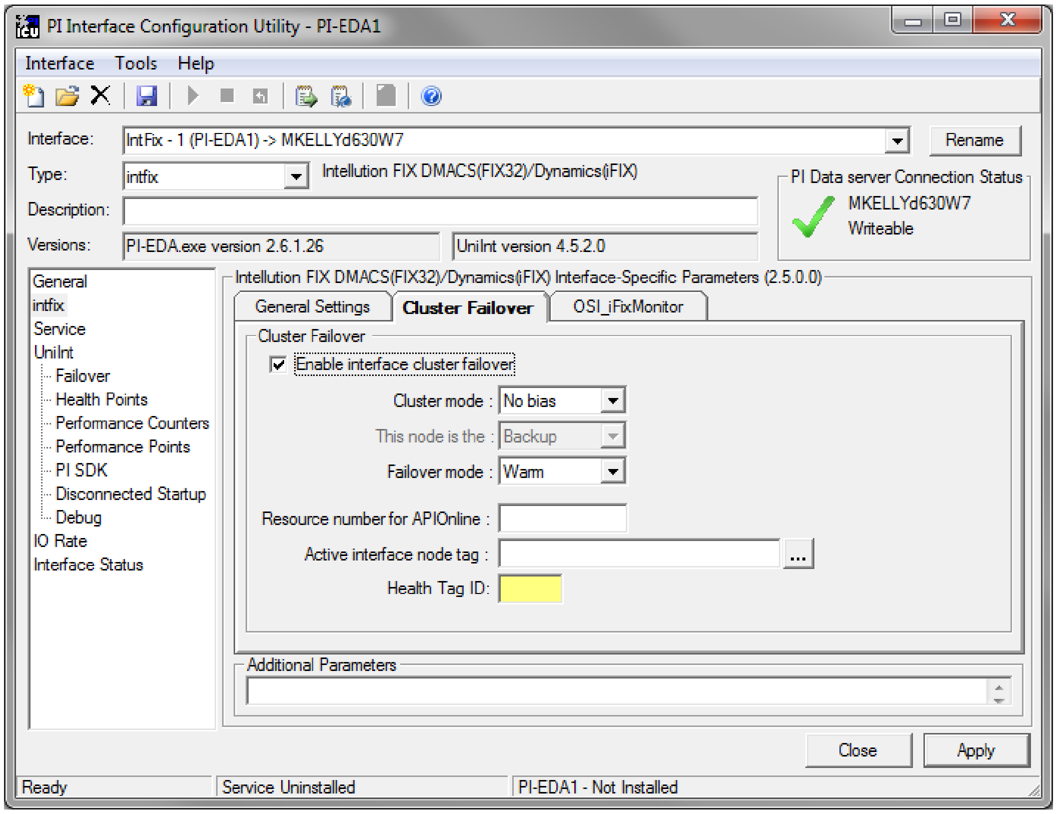
Message Length: 132

Current Length: 130

Using the preceding illustration of the **Common Message Format Configuration** dialog box as an example, note the starting string position of the Value column is 68 and the Value string length is 13. The starting string position is calculated by adding the string length for Date, Time, Node, Tagname, and Alarm Type. The **Column Order** list does not change this calculation as the interface receives alarm/event messages with columns in the order specified in the **Columns** area; the **Column Order** list is of no consequence for the interface (`/C=`*`start:length`*).

Cluster Failover

The interface supports two forms of redundancy. One redundancy option is based on Microsoft Cluster server. See Appendix D: Cluster Failover for a complete discussion on operational requirements and configurations.



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix — Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Description:
Versions: PI-EDA.exe version 2.6.1.26 — UniInt version 4.5.2.0
PI Data server Connection Status: MKELLYd630W7 Writeable

Intellution FIX DMACS(FIX32)/Dynamics(iFIX) Interface-Specific Parameters (2.5.0.0)

Cluster Failover
[✓] Enable interface cluster failover
Cluster mode : No bias
This node is the : Backup
Failover mode : Warm
Resource number for APIOnline :
Active interface node tag :
Health Tag ID:

Additional Parameters

Ready | Service Uninstalled | PI-EDA1 - Not Installed

Enable interface cluster failover

Select this check box to use cluster failover. Upon enabling this box, the failover configuration options will become active (**`/FO`**).

Cluster Mode:

The interface has the ability to operate with a preference for running on a specified cluster node if at all possible. This is referred to as running with Primary bias. This behavior may be preferred if one of the cluster nodes has proven to be more stable or otherwise performs better than the others.

If **Primary bias** is selected in the Cluster mode list, then the **This node is the** option will be enabled. In this box, you must select whether this cluster node is the primary or backup. Note it is critical that only one cluster node be specified as the primary. If more than one cluster node is specified as the primary node, they will compete for ownership of the cluster group resource, sending the interface into an endless loop of failovers.

A Cluster mode of No bias means the interface does not attempt to control which node runs the active interface. As a result, whichever node owns the cluster resource on startup will be the active interface. This will remain so until there is a problem that causes failover or a user uses the Cluster Manger to intentionally manipulate the configuration. The default value for this option is No Bias (**`/CM=0`** for Primary and **`/CM=1`** for No Bias, default=1).

This Node is the:

This option is enabled when Primary bias is specified for the Cluster Mode. In this box, select whether the current node is the primary or backup node for failover operation. The default value for this option is Backup (**`/PR=0`** for primary and **`/PR=1`** for backup, Default:1).

Failover Mode:

The interface has the option of running in either warm or hot failover mode. This behavior determines whether or not an interface running as a backup will query Intellution for point updates.

Warm failover mode means the interface does not query for point updates when operating as the backup node. Hot failover mode tells the interface it should query Intellution for point updates at all times but send them only when active.

Running in hot failover mode has the advantage of minimizing the risk of missing data on failover. However, to minimize loading on inactive cluster nodes, running in warm failover mode is recommended. The default value for this option is Warm (**`/FM=0`** for hot and **`/FM=1`** for warm, Default:1)

Resource number for APIOnline:

The resource number is used to indicate the name of the apionline cluster group resource for the interface. This number will be appended to 'apionline' and used for initialization on interface startup. For example, if a value of 1 is entered, the interface will look for apionline1 as the cluster group resource. A negative number tells the interface that the resource has been defined as simply apionline. A procedure for creating cluster group resources can be found in the section Group and Resource Creation Using Cluster Administrator (**`/RN=#`**).

Active Interface node point:

A PI string point can be specified to receive the name of the node where the active interface is running. The button to the right of this option can be used to launch a PI point search for selecting the desired point (**`/CN=`***`<tagname>`*).

The active cluster point should be configured as follows:

Attribute	**Value**
PointSource	L
PointType	string
Compressing	0
ExcDev	0

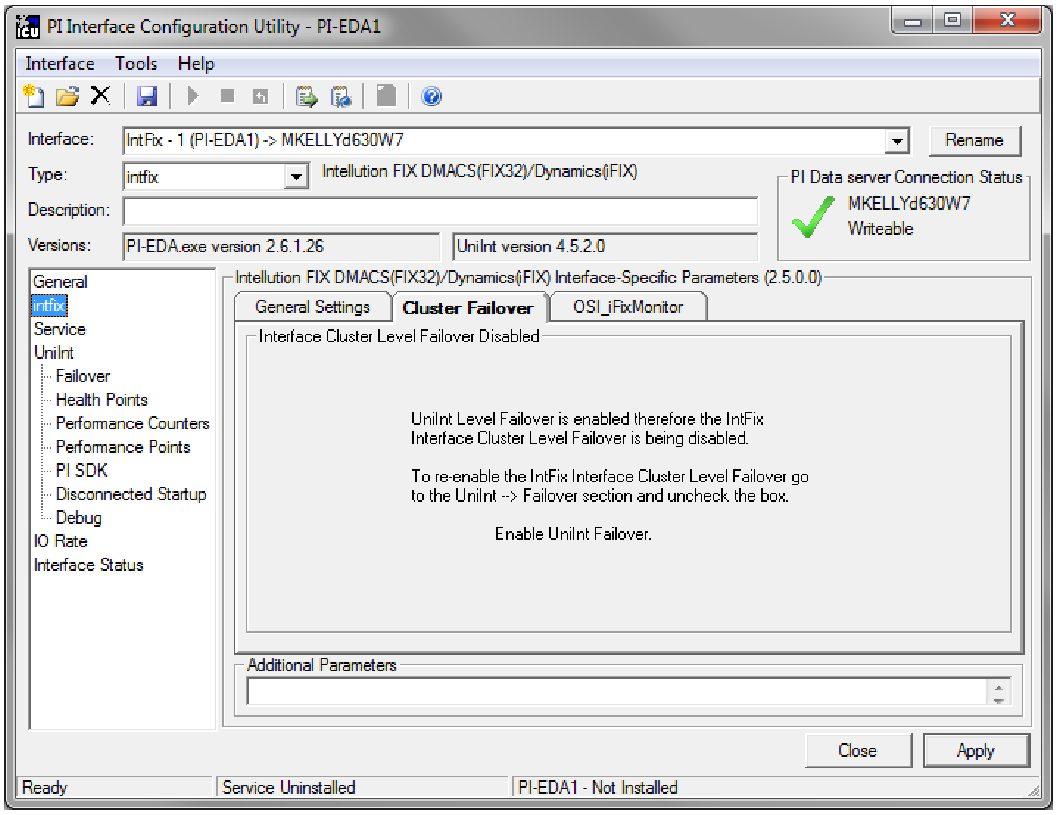
In addition to receiving the name of the active interface node, this point will also receive shutdown events whenever the interface is stopped on any of the cluster nodes. The shutdown event will also contain the name of the machine in the following format: **`Shutdown hostname`**

Health Point ID

This value is used when creating UniInt health points for an interface that uses Non-UniInt interface failover. It is used for the Location3 point attribute for UniInt health points. (`/UHT_ID=#`)

UniInt Failover Enabled

If UniInt Failover is enabled, the following screen will appear when the Cluster Failover tab is selected:



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7

Type: intfix — Intellution FIX DMACS(FIX32)/Dynamics(iFIX)

Versions: PI-EDA.exe version 2.6.1.26 — UniInt version 4.5.2.0

PI Data server Connection Status: MKELLYd630W7 Writeable

Intellution FIX DMACS(FIX32)/Dynamics(iFIX) Interface-Specific Parameters (2.5.0.0)

Interface Cluster Level Failover Disabled

UniInt Level Failover is enabled therefore the IntFix Interface Cluster Level Failover is being disabled.

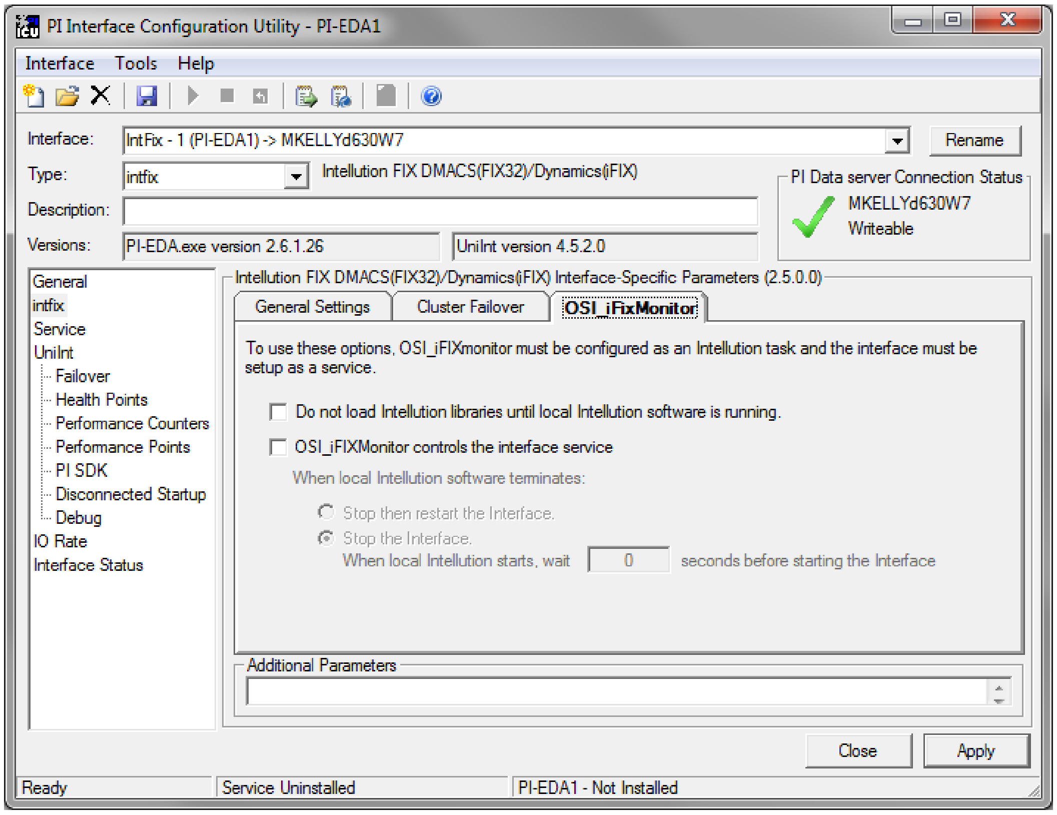
To re-enable the IntFix Interface Cluster Level Failover go to the UniInt --> Failover section and uncheck the box.

Enable UniInt Failover.

Ready | Service Uninstalled | PI-EDA1 - Not Installed

OSI_iFixMonitor

To use these options, OSI_iFixMonitor must be configured as an Intellution task (see section Configuring OSI_iFIXmonitor Program) and the interface must be setup as a service.



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7
Type: intfix — Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
PI Data server Connection Status: MKELLYd630W7 Writeable
Versions: PI-EDA.exe version 2.6.1.26 | UniInt version 4.5.2.0

Intellution FIX DMACS(FIX32)/Dynamics(iFIX) Interface-Specific Parameters (2.5.0.0)

To use these options, OSI_iFIXmonitor must be configured as an Intellution task and the interface must be setup as a service.

- [ ] Do not load Intellution libraries until local Intellution software is running.
- [ ] OSI_iFIXMonitor controls the interface service

When local Intellution software terminates:
- ○ Stop then restart the Interface.
- ● Stop the Interface.
When local Intellution starts, wait 0 seconds before starting the Interface

Ready | Service Uninstalled | PI-EDA1 - Not Installed

Do not load Intellution libraries until local Intellution software is running

Select Do not load Intellution libraries until local Intellution software is running to configure the PI IntFix Interface to verify that Intellution (iFIX) is running on the local node before loading the Intellution libraries. When this option is selected and the PI IntFix Interface starts before iFIX, the Intellution libraries will not be loaded and, therefore, the PI IntFix Interface will not prevent iFIX from starting (**`/DelayLoadEDA`**).

OSI_iFixMonitor controls the interface service

Checking this box puts OSI_iFixMonitor in control of the interface service and enables the configuration of the OSI_iFixMonitor program. There are two choices to pick from which dictate how the interface will be controlled.

Stop then restart the Interface

The **Stop then restart the Interface** option configures OSI_iFIXmonitor to manage the PI IntFix Interface service. When iFIX starts, iFIX starts OSI_iFIXmonitor, which starts the PI IntFix Interface service (if it is not already running). When iFIX stops, it signals OSI_iFIXmonitor to terminate and OSI_iFIXmonitor stops the PI IntFix Interface service, waits for the PI IntFix Interface service to actually terminate, then restarts the PI IntFix Interface service.

Stop the Interface

The **Stop the interface** option configures OSI_iFIXmonitor to manage the PI IntFix Interface service. When iFIX starts, iFIX starts OSI_iFIXmonitor, which starts the PI IntFix Interface service (if it is not already running). When iFIX stops, it signals OSI_iFIXmonitor to terminate and OSI_iFIXmonitor stops the PI IntFix Interface service. The PI IntFix Interface service remains stopped until iFIX is restarted.

When local Intellution starts, wait # seconds before starting the Interface

When using the Stop the Interface option, a wait time can be entered for the number of seconds to wait after iFIX starts before starting the interface. The default is not to wait but start the interface immediately once the iFIX software is running.

Additional Parameters

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

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

Command-line Parameters

Note: The *PI Universal Interface (UniInt) User Guide* includes details about other command-line parameters, which may be useful.

Parameter	Description
`/c=start:length` Optional *Used in conjunction with **`/em`**	Designate the position of data within the alarm/event string. The start value is 1 based.
`/CacheMode` Required Default: **`Not Defined`**	Required for disconnected startup operation. If defined, the **`/CacheMode`** startup parameter indicates that the interface will be configured to utilize the disconnected startup feature.

`/CachePath=path`
Optional
Default: **`Not Defined`**
	Used to specify a directory in which to create the point caching files. The directory specified must already exist on the target machine. By default, the files are created in the same location as the interface executable.
	If the path contains any spaces, enclose the path in quotes.
	Examples:
	`/CachePath=D:\PIPC\Interfaces\CacheFiles`
	`/CachePath=D:/PIPC/Interfaces/CacheFiles`
	`/CachePath=D:/PIPC/Interfaces/CacheFiles/`
	
	Examples with space in path name:
	`/CachePath="D:\Program Files\PIPC\MyFiles"`
	`/CachePath="D:/Program Files/PIPC/MyFiles"`
	`/CachePath="D:/Program Files/PIPC/MyFiles/"`

`/CacheSynch=#` Optional Default: 250 ms	**NOTE:** Care must be taken when modifying this parameter. This value must be less than the smallest scan class period defined with the **`/f`** parameter. If the value of the **`/CacheSynch`** parameter is greater than the scan class value, input scans will be missed while the point cache file is being synchronized. The optional **`/CacheSynch=#`** startup parameter specifies the time slice period in milliseconds (ms) allocated by UniInt for synchronizing the interface point cache file with the PI Data Archive. By default, the interface will synchronize the point cache if running in the disconnected startup mode. UniInt allocates a maximum of # ms each pass through the control loop synchronizing the interface point cache until the file is completely synchronized. Synchronization of the point cache file can be disabled by setting the value **`/CacheSynch=0`**. The minimum synchronization period when cache synchronization is enabled is 50ms Whereas, the maximum synchronization period is 3000ms (3s). Period values of 1 to 49 will be changed by the interface to the minimum of 50ms and values greater than 3000 will be set to the maximum interval value of 3000ms. Default: 250 ms Range: {0, 50 – 3000} time in milliseconds Example: **`/CacheSynch=50`** (use a 50ms interval) **`/CacheSynch=3000`** (use a 3s interval) **`/CacheSynch=0`** (do not synchronize the cache)

Parameter	Description
`/cm=#` Optional *Used in conjunction with **`/fo`** Default**`=1`**	Cluster mode, used for cluster failover. Specifies whether the interface has a bias toward running on the primary node (**`/CM=0`**) or no bias (**`/CM=1`**).
`/cn=tagname` Optional *Used in conjunction with **`/fo`**	When cluster failover is enabled, a PI Data Archive string point can be specified to receive the name of the node where the active interface is running. In addition to receiving the name of the active interface node, this point will also receive shutdown events whenever the interface is stopped on any of the cluster nodes. The shutdown event will also contain the name of the machine in the following format: **`Shutdown hostname`**.

`/db=#`
or
`/db=#,#,...`
Optional
	The interface has the option to enable debug messaging for specific operations.
	Debug options:
	1 – Maximum debug message level.
	2 – Point checking on startup and point edits. Note this will slow interface startup proportional to the number of points (more points means slower startup).
	3 – Input data.
	4 – Output data.
	5 – Alarm/event message data collection.
	6 – Cluster failover.

`/DelayLoadEDA` Optional	When the interface is installed on an iFIX node, once the interface loads the EDA library and calls it, the EDA library acquires resources whose existence will prevent iFIX from starting if it is not already running. If this parameter is not used, when the interface starts, the interface loads the EDA library and begins calling it. Consequently, if the interface starts before iFIX, iFIX will refuse to start. The **`/DelayLoadEDA`** parameter prevents the interface from loading or calling the EDA library until iFIX is verified to be running. Therefore, if the interface starts before iFIX, the interface will not prevent iFIX from starting. **Note**: For the interface to verify that iFIX is running without using the EDA library, the OSI_iFIXmonitor program must be configured in iFIX as a task that iFIX starts. OSI_iFIXmonitor is only needed if the **`/DelayLoadEDA`** parameter is used. After the interface detects that iFIX is running, it loads and begins using the EDA library. Once acquired, the resources used by the EDA library cannot be released dynamically. If iFIX stops, iFIX will refuse to restart until the interface terminates, which releases the EDA library resources. See the companion parameter **`/StopWithFIX`**.

Parameter	Description
/ec=# Optional	The first instance of the **/ec** parameter on the command-line is used to specify a counter number, #, for an I/O Rate point. 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, every interface that is running without **/ec=**# explicitly defined will write to the same I/O Rate point. Either explicitly define an event counter other than 1 for each instance of the interface or do not associate 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. For interfaces that run on Windows nodes, subsequent instances of the **/ec** parameter may be used by specific interfaces to keep track of various input or output operations. Subsequent instances of the **/ec** parameter can be of the form **/ec***, where * is any ASCII character sequence. For example, **/ecinput=10**, **/ecoutput=11**, and **/ec=12** are legitimate choices for the second, third, and fourth event counter strings.
/em Optional	Enable data collection for alarm/event messages. When specified, all points belonging to scan class 1 will be used to record alarm data on a point for point basis. In addition, the interface can be configured to send all alarm/event messages to a single PI string point (**/al=***tagname*). The WUSERQ used for alarm/event data collection is specified using the **/qn** switch.
/f=SS.## or **/f=SS.##,SS.##** or **/f=HH:MM:SS.##** or **/f=HH:MM:SS.##, hh:mm:ss.##** Required for reading scan-based inputs	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 the interface; the second occurrence defines the second scan class, and so on. PI Points are associated with a particular scan class via the Location4 PI Point attribute. For example, all PI 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

Parameter	Description

	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 and/or UNIX. 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.

`/fm=#` Optional *Used in conjunction with **`/fo`** Default**`=1`**	The interface has the option of running in either warm or hot failover mode. This behavior determines whether or not an interface running as a backup will query Intellution for point updates. Warm failover mode means the interface does not query for point updates when operating as the backup node. Hot failover mode tells the interface it should query Intellution for point updates at all times but send them only when active. The advantage of running in hot failover mode is that you minimize the risk of missing data on failover. However, to minimize loading on inactive cluster nodes, OSIsoft recommends running in warm failover mode. The default value for this option is Warm. 0 -> Hot 1 -> Warm
`/fo` Optional	Enables cluster failover support. A complete discussion on failover operation and configuration can be found in Appendix D: Cluster Failover.
`/h`	Running the interface from a command prompt with **`/h`** as the only parameter causes the interface to print its version and a list of parameters – essentially an on line summary of this table.
`/help or /?`	Running the interface from a command prompt with **`/help`** or **`/?`** As the only parameter causes UniInt to print its version, a list of UniInt Service configuration parameters, and a list of UniInt generic interface parameters.

Parameter	Description

`/host=host:port`
Required
	The **`/host`** parameter specifies the PI Data Archive node. *Host* is the IP address or the domain name of the PI Data Archive node. `Port` is the port number for TCP/IP communication. The port is always 5450. It is recommended to explicitly define the host and port on the command-line with the **`/host`** parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults.
	
	Examples:
	
	The interface is running on an interface node, the domain name of the PI Data Archive 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` Highly Recommended	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`**
`/ls` Optional	The default behavior of the interface is to use the PI Data Archive system time for the data timestamp. Use **`/ls`** to specify that the interface should use the local interface node system time for timestamp source. This option must be used with caution. If the local system time is ahead of the PI Data Archive, the data may be rejected. PI discards "future" data, which is defined as any event more than 10 minutes ahead of the PI Data Archive time. This option should only be used if there is a compelling reason to do so.
/pr=# Optional *Used in conjunction with **`/fo`** and **`/cm=0`** Default=1	When cluster failover is enabled and the cluster mode is set for primary bias, this switch is used to designate the local node as primary (**`/pr=0`**) or backup (**`/pr=1`**).
`/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 length of **`X`** is limited to 100 characters by UniInt. **`X`** can contain any character except '*' and '?'. The point source that is assigned with the **`/ps`** parameter corresponds to the PointSource attribute of individual PI Points. The interface will attempt to load only those PI points with the appropriate point source. If the PI API version being used is prior to 1.6.x or the PI Data Archive version is prior to 3.4.370.x, the `PointSource` is limited to a single character unless the SDK is being used.

Parameter	Description
`/qn=#` Optional *Used in conjunction with **`/em`** Default: 1	When alarm/event message data collection is enabled, this switch is used to specify whether WUSERQ1 (**`/qn=1`**) or WUSERQ2 (**`/qn=2`**) is used for the data source.
`/rn=#` Optional *Used in conjunction with **`/fo`**	The resource number is used to indicate the name of the apionline cluster group resource for the interface. This number will be appended to 'apionline' and used for initialization on interface startup. For example, if you enter a value of 1, the interface will look for apionline1 as the cluster group resource. A negative number tells the interface that the resource has been defined as simply apionline. A procedure for creating cluster group resources can be found in the section Group and Resource Creation Using Cluster Administrator.
`/sio` Optional	The **`/sio`** parameter stands for "suppress initial outputs." The parameter applies only for interfaces that support outputs. If the **`/sio`** parameter is not specified, the interface will behave in the following manner. When the interface is started, the interface determines the current Snapshot value of each output point. Next, the interface writes this value to each output point. In addition, whenever an individual output point is edited while the interface is running, the interface will write the current Snapshot value to the edited output point. This behavior is suppressed if the **`/sio`** parameter is specified on the command-line. That is, outputs will not be written when the interface starts or when an output point is edited. In other words, when the **`/sio`** parameter is specified, outputs will only be written when they are explicitly triggered.

Parameter	Description
`/stopstat=digstate` or **`/stopstat`** **`/stopstat`** only is equivalent to **`/stopstat="Intf Shut"`** Optional Default = no digital state written at shutdown.	If **`/stopstat=digstate`** is present on the command line, then the digital state, `digstate`, will be written to each PI Point when the interface is stopped. For a PI Data Archive, `digstate` must be in the system digital state table. . UniInt will use the first occurrence of **`digstate`** found in the table. If the **`/stopstat`** parameter is present on the startup command line, then the digital state "`Intf Shut`" will be written to each PI Point when the interface is stopped. 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 PI Point when the interface is stopped. This prevents the digital state being written to PI Points while a redundant system is also writing data to the same PI 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`.
`/StopWithFIX` **`Optional`**	When the interface is installed on an iFIX node, once the interface loads the EDA library and calls it, the EDA library acquires resources whose existence will prevent iFIX from starting if it is not already running. Once acquired, the resources used by the EDA library cannot be released dynamically and are only released when the interface terminates. If iFIX stops after the interface begins calling the EDA library, iFIX will refuse to restart until all EDA client programs, including the interface, terminate and consequently release the EDA library resources. The **`/StopWithFIX`** parameter causes the interface to terminate itself when it detects that iFIX has changed from running to stopped. When the interface terminates, the EDA library resources are released, so iFIX will not be prevented from restarting. **Note** that the interface does not simply check for iFIX not running; iFIX must transition from running to stopped for the interface to self-terminate. Otherwise, the interface would immediately terminate if it were started before iFIX. When the interface is (or can be) started before iFIX, the companion parameter /DelayLoadEDA must also be used to prevent the interface from using the EDA library before iFIX starts. **Note**: This parameter causes the interface to terminate. If the interface is configured as a Windows service (the usual case), do not use this parameter. Instead, configure the OSI_iFIXmonitor program to control the interface service. See section Configuring the Interface with PI ICU or Appendix F: OSI_iFIXmonitor Program.

Parameter	Description
`/UFO_ID=#` Required for UniInt failover phase 1 or 2	Failover ID. This value must be different from the Failover ID of the other interface in the failover pair. It can be any positive, non-zero integer.
`/UFO_Interval=#` Optional Default: 1000 for phase 1 failover Default: 5000 for phase 2 failover Valid values are 50-20000.	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 points as well as how often UniInt checks on the status of the other copy of the interface.
`/UFO_OtherID=#` Required for UniInt failover phase 1 or 2	Other Failover ID. This value must be equal to the Failover ID configured for the other interface in the failover pair.
`/UFO_Sync=path/[filename]` Required for UniInt Interface Level Failover Phase 2 synchronization. Any valid pathname / any valid filename The default filename is generated as *executablename_pointsource_interfaceID.dat*	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 d 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 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 for UniInt failover phase 2.	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 shut down and log an error to the log file stating the requested failover type is not supported.

Parameter	Description
/uht_id=# Optional Required if any type of failover other than UniInt failover phase 1 or 2 is supported.	The **/uht_id=#** command-line parameter is used to specify a unique ID for interfaces that are run in a redundant mode without using the UniInt failover mechanism. There are several OSIsoft interfaces that are UniInt based and implement their own version of failover. In order for health point(s) to be configured to monitor a single copy of the interface, an additional parameter is required. If the **/uht_id=#** is specified, only health points with a location3 value equal to **#** will be loaded.
/w=# Optional Default: 120	This specifies how many seconds the interface waits on startup before connecting to Intellution allowing it to fully start.

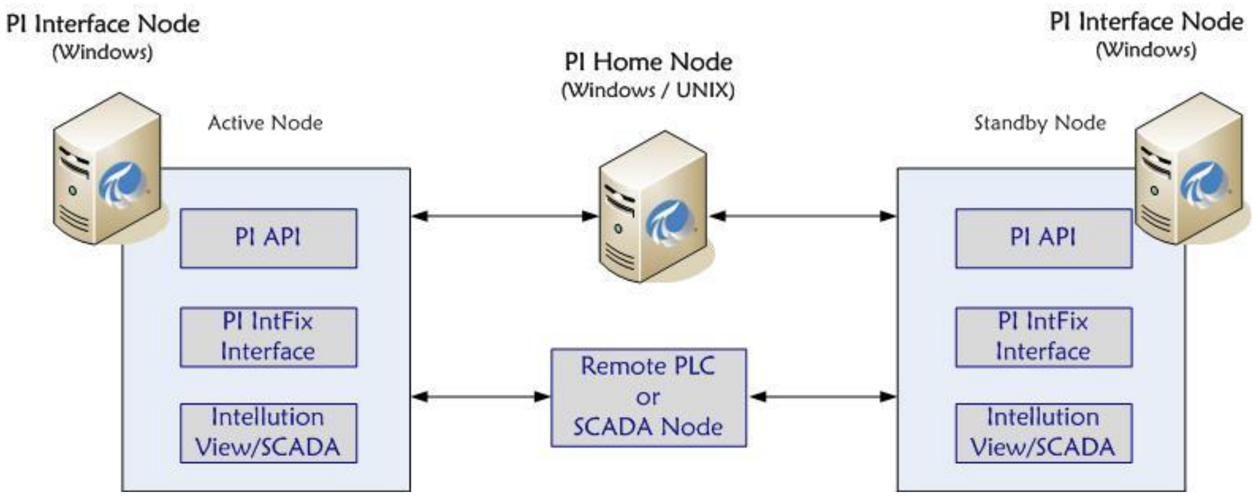
Sample PI-EDA.bat File

The following is an example file:

```
REM=============================================================
REM
REM PI-EDA.bat
REM
REM Sample startup command file for the
REM Intellution Fix DMACS (FIX32) / Dynamics (iFIX)
REM
REM ===========================================================
REM
REM OSIsoft strongly recommends using PI ICU to modify startup files.
REM
REM Sample command line
REM
    .\PI-EDA.exe ^
    /host=XXXXXX:5450 ^
    /ps=IntFix ^
    /id=1 ^
    /w=120 ^
    /ec=2 ^
    /f=00:00:01 ^
    /em ^
    /c=67:13
REM
REM End of PI-EDA.bat File
```

Chapter 10. 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: Although both failover methods successfully maintain continuous data flow OSIsoft recommends using Phase 2 because it is supported by more interfaces.

Phase 1 is appropriate in only two situations: (1) if performance degradation occurs using the shared file or (2) read/write permissions for the shared file cannot be granted to both interfaces.

You can also configure UniInt failover to send data to a High Availability (HA) PI Data Archive collective. The collective provides redundant PI Data Archives to allow for the uninterrupted collection and presentation of time series data. In an HA configuration, PI Data Archives can be taken down for maintenance or repair. The HA PI Data Archive collective is described in the *High Availability Administrator Guide*.

When configured for UniInt failover, the interface routes all PI point data through a state machine. The state machine determines whether to queue data or send it directly to a PI point depending on the current state of the interface. When the interface is in the active state, data sent through the interface is routed directly to a PI point. In the backup state, data from the interface is 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 PI data from the data source.
- Interface 2 is the Backup interface for collection of PI data from the data source.
- Phase 1: The data source must be configured with six failover control points (input and output points for three failover control types):

  (1) Active ID.

  (2) Heartbeat for Interface 1.

  (3) Heartbeat for Interface 2.
- You must set up a shared file if using Phase 2 failover..
- Phase 2: The shared file must store data for five failover control points:

  (1) Active ID.

  (2) Heartbeat 1.

  (3) Heartbeat 2.

  (4) Device Status 1.

  (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 PI Data Archive host for exceptions and PI point updates.
- All other configuration parameters for the two interfaces must be identical.

Synchronization through the Data Source (Phase 1)

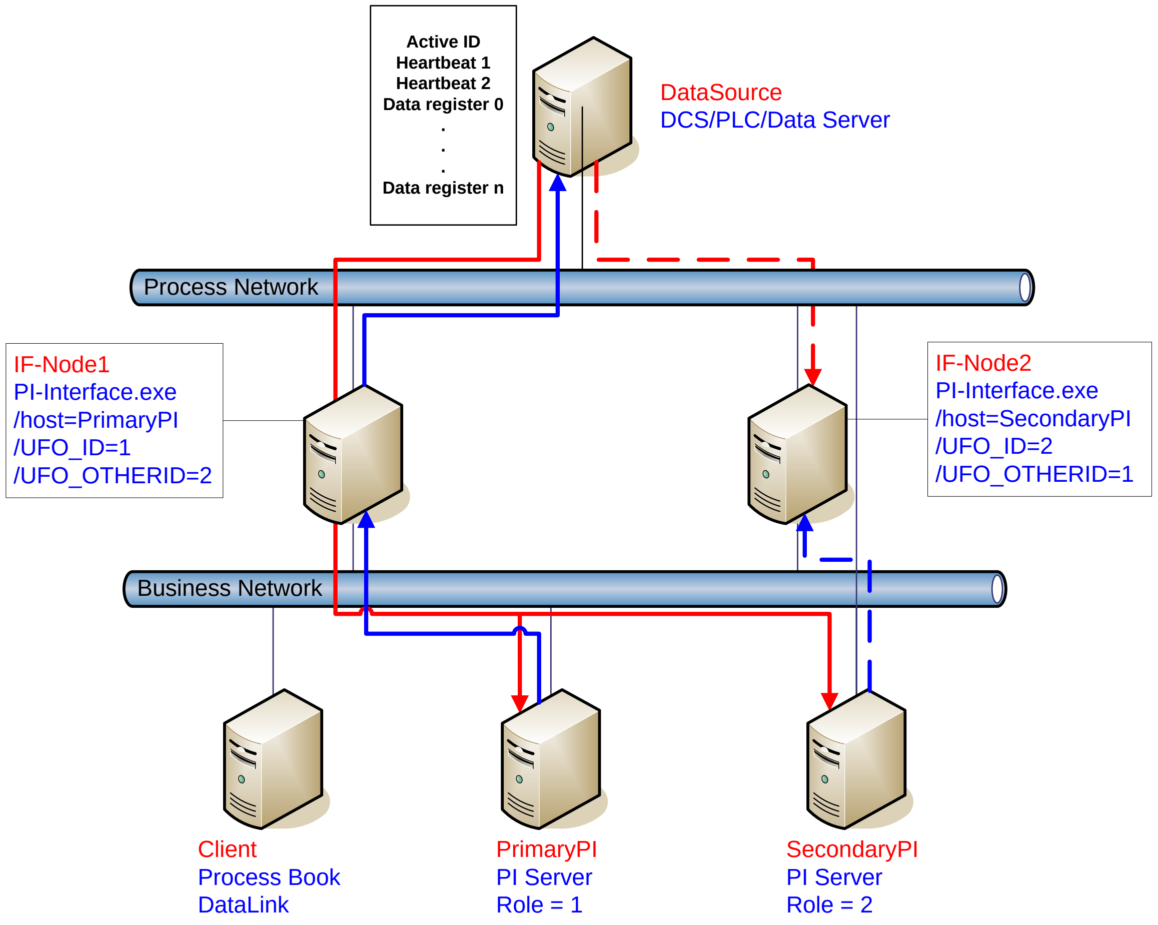


Figure 1: Failover Architecture Phase 1 – Synchronization through the Data Source

Figure 1 shows Phase 1 failover architecture. The diagram shows a typical network setup. This by no means represents the myriad possible network configurations; it is an example only for the following discussions. This example is explained in greater detail after the discussion of the start-up parameters, data source points, and PI points.

Configuring Synchronization through the Data Source (Phase 1)

Step	Description
1.	Verify non-failover interface operation as described in the Installation Checklist section of this manual
2.	**Configure Points on the Data Source** Create three points (Active ID, Heartbeat 1 and Heartbeat 2) on the data source. The interface must be able to read from and write to these points. The **ActiveID** must accept values from 0 to the highest failover ID. The two heartbeat points must accept values from 0 to 31. See the Data Source Points section below.
3.	**Use the Interface Configuration Utility to configure the interface parameters** Enable failover by selecting "Enable UniInt Failover" in the Failover section of the Interface Configuration Utility (ICU) and assign the appropriate number for the two Failover IDs: (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. All other command line parameters for the Primary and Backup interfaces must be identical. If you are using a PI Data Archive collective, you must specifically identify the primary and backup interfaces as different members of the collective. [Optional] Set the update rate for the heartbeat point if the input points are unsolicited.
4.	**Configure the PI points** You must configure six PI points for the interface (input and output points for each of the three failover points on the data source). For more information about configuring input and output points, refer to the PI Point Configuration chapter You can also configure two state points for monitoring the status of the interfaces.

Tag	ExDesc	digitalset	The remaining attributes must be configured according to the PI Point Configuration chapter so the PI points map to the correct points on the data source
ActiveID_In	[UFO_ACTIVEID]		
ActiveID_Out	[UFO_ACTIVEID]		
IF1_HB_In (IF-Node1)	[UFO_HEARTBEAT:#]		
IF1_HB_Out (IF-Node1)	[UFO_HEARTBEAT:#]		
IF2_HB_In (IF-Node2)	[UFO_HEARTBEAT:#]		
IF2_HB_Out (IF-Node2)	[UFO_HEARTBEAT:#]		
IF1_State (IF-Node1)	[UFO_STATE:#]	IF_State	
IF2_State (IF-Node2)	[UFO_STATE:#]	IF_State	

5.	If using PI APS versions earlier than 1.2.4.0 to synchronize the Data Source and PI points, special attention must be paid to the failover control points. Check that the failover control points are not included in the PI APS synchronization scheme. Synchronizing the control points will cause the failover points to be edited by PI APS and may result in possible interface shutdown.
6.	**Test the configuration.** Run the interface with the six Failover Control PI points to ensure their proper operation.

Step	Description
	1. Start the primary interface interactively without buffering. 2. Verify a successful interface start by reviewing the 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 PI. 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. For example, verify data on the FIX32/iFix node using the Database Builder Utility. • The Active ID control point on the FIX32/iFix node 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 point on the FIX32/iFix node must be changing values at a rate specified by the `/UFO_Interval` startup command-line parameter. 4. Verify data on the PI Data Archive using available PI tools. • The Active ID control point on the PI Data Archive 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 point on the PI Data Archive must be changing values at a rate specified by the **`/UFO_Interval`** startup command-line parameter. 5. Stop the primary interface. 6. Start the backup interface interactively without buffering. Notice that this copy will become the primary because the other copy is stopped. 7. Repeat steps 2, 3, 4 and 5. 8. Stop the backup interface. 9. Start buffering. 10. Start the primary interface interactively. 11. Once the primary interface has successfully started and is collecting data, start the backup interface interactively. 12. Verify that both copies of the interface are running in a failover configuration. • Review the 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 PI. 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 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. 13. For example: verify data on the FIX32/iFix node using the Database Builder utility. • The Active ID control point on the FIX32/iFix node must be set to the

Step	Description
	value of the running copy of the interface that was started first as defined by the **/UFO_ID** startup command-line parameter. • The Heartbeat control points for both copies of the interface on the FIX32/iFix node must be changing values at a rate specified by the **/UFO_Interval** startup command-line parameter. 14. Verify data on the PI Data Archive using available PI tools. • The Active ID control point on the PI Data Archive must be set to the value of the running copy of the interface that was started first as defined by the **/UFO_ID** startup command-line parameter. • The Heartbeat control points for both copies of the interface on the PI Data Archive 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. 15. Test Failover by stopping the primary interface. 16. Verify the backup interface has assumed the role of primary by searching the 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 PI. Backup interface not available.`" The backup interface is now considered primary and the previous primary interface is now backup. 17. Verify no loss of data in the PI Data Archive. There may be an overlap of data due to the queuing of data. However, there must be no data loss. 18. 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 PI. Backup interface available.`" In the log file. 19. 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 log file. 20. Test failover with different failure scenarios (e.g. loss of PI Data Archive 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 the PI Data Archive and on the data source. 21. Stop both copies of the interface, start buffering, start each interface as a service. 22. Verify data as stated above. 23. 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 the Data Source (Phase 1)

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 PI Point when the interface is stopped. This prevents the digital state being written to PI Points while a redundant system is also writing data to the same PI 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. All of the parameters are required except the **`/UFO_Interval`** startup parameter.

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_Interval=#`	Optional	Failover Update Interval Specifies the update Interval in milliseconds and must be the same on both interface computers. This is the rate at which UniInt updates the failover heartbeat points as well as how often UniInt checks on the status of the other copy of the interface. The **`/UFO_Interval`** is only used if the failover control input PI points are collected on an unsolicited basis. If the input PI points are scanned, the failover update interval is determined by the scan class associated with the points.	50 – 20000 / **1000**

Parameter	Required/ Optional	Description	Value/Default
`/Host=server`	Required	Host PI Data Archive for Exceptions and PI point updates The value of the `/Host` startup parameter depends on the PI Data Archive configuration. If the PI Data Archive 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 PI Data Archive collective, the value of the `/Host` parameters on the different interface nodes must point to different members of the collective. This configuration ensures that outputs continue to be sent to the Data Source if one of the PI Data Archives becomes unavailable for any reason.	For IF-Node1 PrimaryPI / **None** For IF-Node2 SecondaryPI / **None**

Data Source Points

The following table identifies the points that are required to manage failover and the values used for each PI attribute.

The following table explains each of the points required on the data source in more detail.

Point	Description	Value / Default
ActiveID	Monitored by the interfaces to determine which interface is currently sending data to the PI Data Archive. **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

PI Points

The following tables list the required UniInt failover control PI points, the values they will receive, and descriptions.

Active_ID Point Configuration

Attributes	ActiveID IN	ActiveID OUT
Tag	<Intf>_Active_IN	<Intf>_Active_OUT
Compmax	0	0
ExDesc	[UFO_ActiveID]	[UFO_ActiveID]
Location1	Match # in **`/id=#`**	Match # in **`/id=#`**
Point Source	Match x in **`/ps=x`**	Match x in **`/ps=x`**
Point Type	Int32	Int32
Shutdown	0	0
Step	1	1

Heartbeat Point Configuration

Attribute	Heartbeat 1 IN	Heartbeat 1 OUT	Heartbeat 2 IN	Heartbeat 2 OUT
Tag	<HB1>_IN	<HB1>_OUT	<HB2>_IN	<HB2>_OUT
ExDesc	[UFO_Heartbeat:#] Match # in **`/UFO_ID=#`**	[UFO_Heartbeat:#] Match # in **`/UFO_ID=#`**	[UFO_Heartbeat:#] Match # in **`/UFO_OtherID=#`**	[UFO_Heartbeat:#] Match # in **`/UFO_OtherID=#`**
Location1	Match # in **`/id=#`**	Match # in **`/id=#`**	Match # in **`/id=#`**	Match # in **`/id=#`**
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 Point Configuration

Attribute	Primary	Backup
Tag	<Tagname1>	<Tagname2>
Compmax	0	0
DigitalSet	UFO_State	UFO_State
ExDesc	[UFO_State:#] (Match **`/UFO_ID=#`** on primary node)	[UFO_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 PI points in more detail.

PI Point ExDesc	**Required / Optional**	**Description**	**Value / Default**
[UFO_ACTIVEID] (Used for both the ActiveID IN and OUT points.)	Required	The active ID input point must be configured as an input PI point for the interface and it must be configured to read the **`ActiveID`** on the data source. Consult the PI Point Configuration chapter for a description of configuring input points. The ExDesc must start with the case sensitive string: [UFO_ACTIVEID]	0 – highest Failover ID / **None** Updated by the redundant Interfaces
[UFO_HEARTBEAT:#] (IF-Node1)	Required	The Heartbeat 1 Output Point must be configured as an output PI point for the interface and it must be configured to write to the Heartbeat 1 point on the data source. Consult the PI Point Configuration chapter for information about configuring output points. The ExDesc must start with the case sensitive string: [UFO_HEARTBEAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node1.	0 – 31 / **None** Updated by the interface on Node 1
[UFO_HEARTBEAT:#] (IF-Node2)	Required	The Heartbeat 2 Input Point must be configured as an input PI point for the interface and it must be configured to read the Heartbeat 2 Point on the Data Source. Consult the PI Point Configuration chapter for information about configuring input points. The ExDesc must start with the case sensitive string: [UFO_HEARTBEAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2.	0 – 31 / **None** Updated by the interface on Node 2

PI Point ExDesc	Required / Optional	Description	Value / Default
[UFO_STATE:#] (IF-Node1)	Optional	The failover state points are optional and do not require a point on the Data Source. The value of the state point can be written to the data source by configuring a normal interface output point and setting the SourceTag attribute to the failover state point. The number following the colon (:) must be the Failover ID for the interface running on IF-Node1. The failover state points are digital points assigned to a digital state set with the following values. 0 = Off: The interface has been shut down. 1 = Backup No Data Source: The interface is running but is unable to communicate to the data source. 2 = Backup No PI Connection: The interface is running and connected to the data source but has lost its communication to the PI Data Archive. 3 = Backup: The interface is running and collecting data normally and is ready to take over if the primary shuts down or experiences problems. 4 = Transition: The interface stays in this state for only a short period of time. The transition period is designed to prevent thrashing when both primary and backup interfaces attempt to assume the role of the primary interface. 5 = Primary: The interface is running, collecting data, and sending the data to the PI Data Archive.	0 – 5 / None Updated by the Primary Interface
[UFO_STATE:#] (IF-Node2)	Optional	The failover state points are optional and do not require a point on the data source. The value of the state point can be written to the data source by configuring a normal interface output point and setting the SourceTag attribute to the failover state point. The number following the colon (:) must be the Failover ID for the interface running on IF-Node2.	0 – 5 / None Updated by the Backup Interface

Detailed Explanation of Synchronization through the Data Source

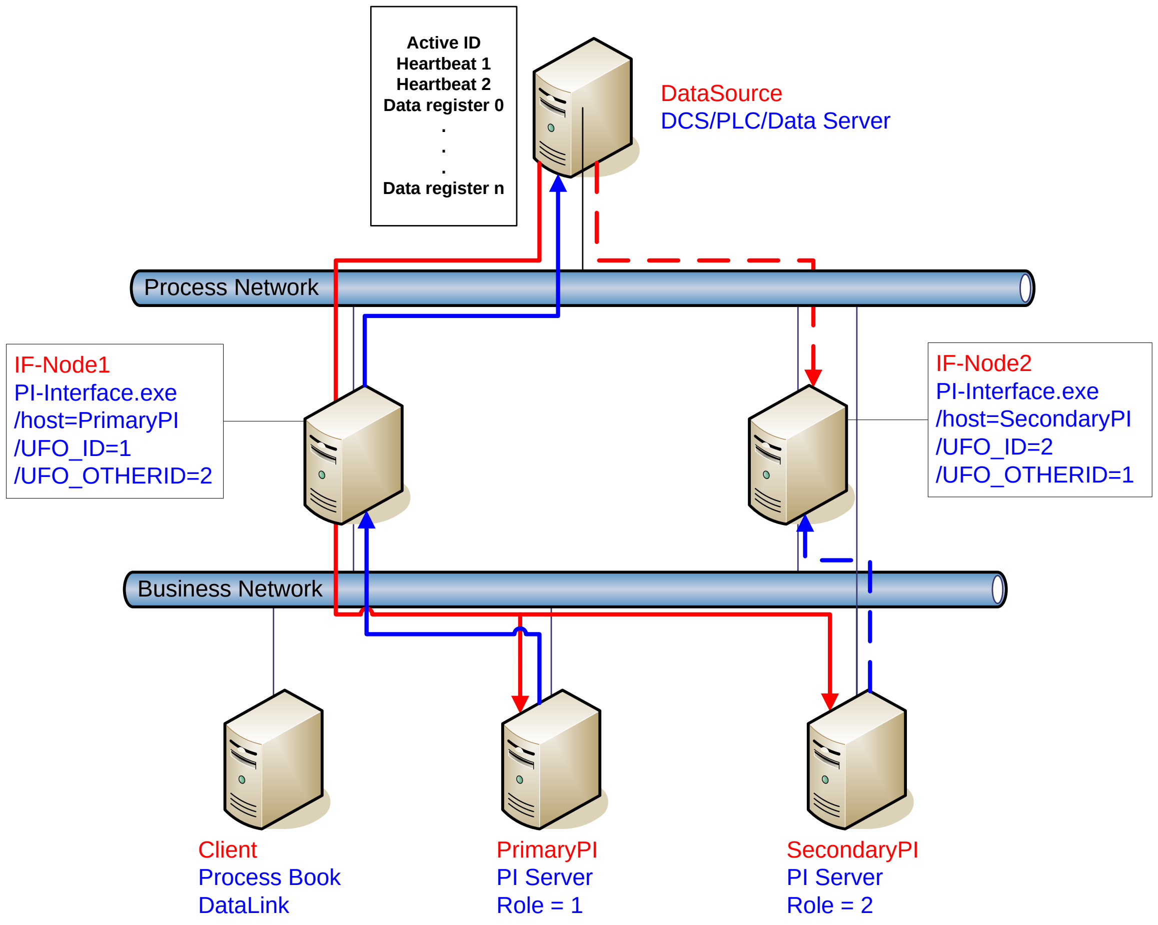


Figure 2 Synchronization through Data Source (Phase 1) Failover Architecture

Synchronization through the data source uses two separate interface nodes communicating with the data source. The failover scheme requires six failover control points in the PI Data Archive and three control points on the data source to control failover operation. The PI points initialize the interface with configuration information for reading and writing to the control points on the data source. Once the interface is configured and running, the ability to read or write to the PI points is not required for failover operation because only the control points on the data source are monitored. However, the PI point values are sent to the PI Data Archive so that you can monitor them with standard OSIsoft client tools. You can force manual failover by changing the active ID point on the data source to the backup failover ID.

The figure above shows a typical network in the normal or steady state. This diagram doesn't represent the myriad configurations that can be supported; it is simply an example for the following discussions. If your hardware configuration differs from the figure, the settings for the Primary and Backup interfaces remain the same with the exception of the `/host` startup parameter. If the interfaces communicate with a stand-alone PI Data Archive, the `/host` parameter for both interfaces must be the same.

To ensure that output to the data source continues when a PI Data Archive in the collective becomes unavailable, the interface running on the primary node (IF-Node1) needs the **`/host`** parameter set to a PI Data Archive that is part of the collective, and the interface running on the backup node (IF-Node2) needs the **`/host`** parameter set to a different PI Data Archive in the same collective.

The continued operation of output when a PI Data Archive becomes unavailable presumes the source data for output data (that is, data read from the PI Data Archive and written to the data source) comes into the PI Data Archive from a process that sends values to all of the PI Data Archives in the collective via n-way buffering.

The solid red line in the figure shows input data flow when the interface on IF-Node1 is in the primary state. The data is read from the data source by the interface and sent to a buffer. Buffering sends the input data to all of the PI Data Archives in the collective via n-way buffering.

The solid blue line shows output data flow. Since the interface on IF-Node1 is configured with **`/host=PrimaryPI`**, the interface signs up for exceptions with the PI Data Archive on PrimaryPI. Exceptions are received by the interface and sent to the data source via the interface.

The dashed red line shows input data flow to the backup interface. The dashed line stops at the interface because the interface does not send the data to buffering unless the interface is in the primary state. If the backup interface transitions to the primary state for any reason, the backup interface begins to send the input data to buffering. Buffering continues to write the data to all of the PI Data Archives in the collective via n-way buffering.

The dashed blue line shows output data flow to the backup interface. The dashed line stops at the interface because an interface does not send data to the data source unless the interface is in the primary state. When the backup interface becomes the primary for any reason, it begins to send output data to the data source.

In the event that the Primary PI Data Archive becomes unavailable for any reason, the primary interface informs the backup interface that it has lost its connection to the PI Data Archive. The backup interface becomes the primary interface because its status is better than the current primary interface. However, if the entire network goes off line and both primary and backup interfaces lose their connection to their respective PI Data Archives, the primary interface remains primary because the current status of the backup interface is the same as the primary, not better. In this case, output data cannot flow to the data source because there is no way for any of the interfaces to get the exception data.

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 the PI Data Archive. The primary interface is also updating its heartbeat point, monitoring the heartbeat point for the backup interface, and checking the **ActiveID** every failover update interval. In this state, the backup interface is actively collecting and queuing data but not sending the received data to the PI Data Archive. It too is updating its heartbeat point, monitoring the heartbeat point for the primary interface, and checking the **ActiveID** every failover update interval. As long as the heartbeat point for the primary interface indicates that it is operating properly and the **ActiveID** has not changed, the backup interface will continue in this mode of operation.

The interaction of the control points is fundamental to failover. The discussion that follows only refers to the data written to the control points on the data source. However, every value written to the control points on the data source is echoed to the control points in the PI Data Archive. Updating of the control points is assumed to take place unless communication with the PI Data Archive is interrupted. The updates to the PI Data Archive will be buffered by Bufserv or PIBufss in this case.

Each interface copy participating in the failover solution queues two 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 2 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 1 second will result in overlapping data between 0 and 2 seconds. The no data loss claim 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 point. In normal operation, the value of the Heartbeat point on the data source is incremented by UniInt from 1 – 15 and then wraps around to a value of 1 again. UniInt increments the heartbeat point on the data source every failover update interval. The default failover update interval is 1 second. UniInt also reads the value of the heartbeat point for the other interface copy participating in failover every failover update interval. If the connection to the PI Data Archive is lost, the value of the heartbeat point increments from 17 – 31 and then wrap around to a value of 17 again. Once the connection to the PI Data Archive is restored, the heartbeat values revert back to the 1 to 15 range. During a normal shutdown process, the heartbeat value is set to zero.

During steady state, the **ActiveID** is the failover ID of the primary interface. This value is set by UniInt when the interface enters the primary state and is not changed by the primary interface until it shuts down gracefully. During shutdown, the primary interface sets the **ActiveID** to zero before shutting down. The backup interface can assume control as primary even if the current primary is not experiencing a problem. You can force this transition by setting the **ActiveID** control point on the data source to the failover ID of the desired interface.

To prevent data loss during failover, the backup interface continuously queues data in memory for the two most recent failover update intervals. As long as the backup interface determines that the primary interface is in good status, the backup interface simply maintains this queue with the most recent data. When the backup interface transitions to primary status, the backup begins transmitting to PI starting with the queued data

Synchronization through a Shared File (Phase 2)

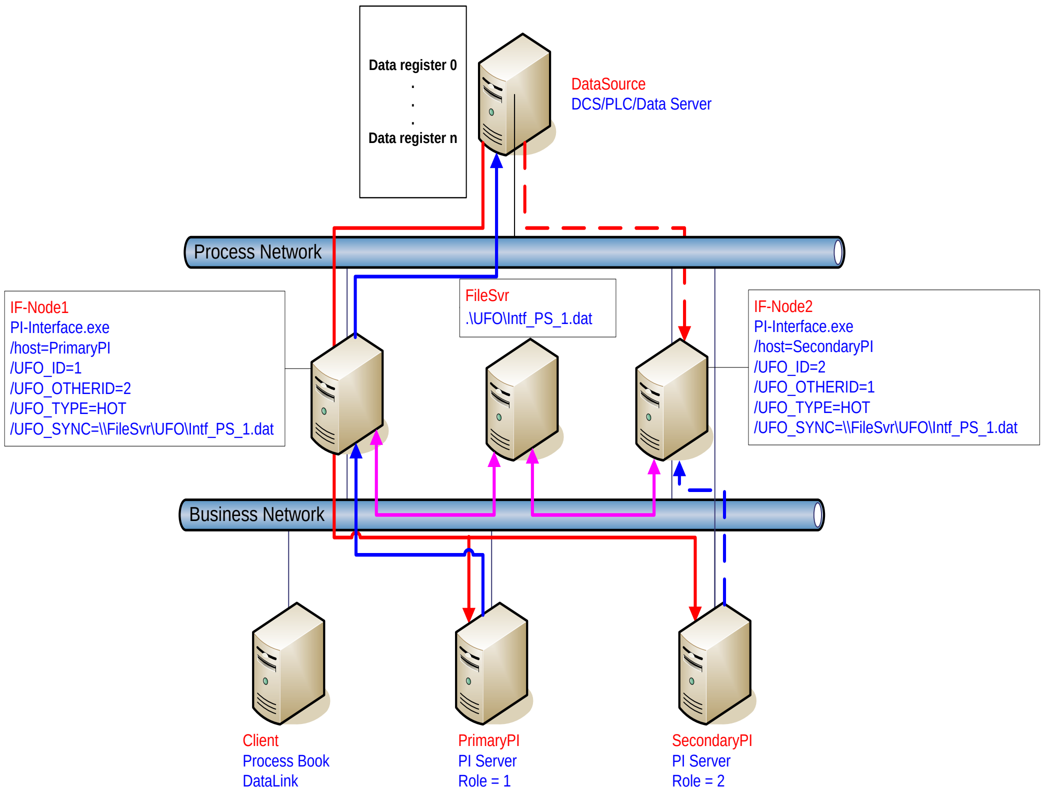


Figure 3: Synchronization through a Shared File (Phase 2) Failover Architecture

The Phase 2 failover architecture is shown in Figure 2 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 or on a separate node from the Interfaces; however OSIsoft strongly recommends that you put the file on a Windows Server platform that has the "File Server" role configured. . Set up 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 select the *Phase 2* option 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* page 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 PI Data Archive 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 PI points** Configure five PI points for the interface: the active ID, heartbeat 1, heartbeat2, device status 1 and device status 2. You can also configure two state points for monitoring the status of the interfaces. Do not confuse the failover device status points with the UniInt health device status points. The information in the two points is similar, but the failover device status points are integer values and the health device status points 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.**

Step	Description
	After configuring the shared file and the interface and PI points, the interface should be ready to run. For help resolving failover file issues, see knowledge base article 889 on the OSIsoft technical support web site. 1. Start the primary interface interactively without buffering. 2. Verify a successful interface start by reviewing the 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 PI. 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 PI Data Archive using available PI tools. • The active ID control point in the PI Data Archive 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 point on the PI Data Archive 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, and 3. 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 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 PI. 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 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 PI Data Archive using available PI tools. • The active ID control point in the PI Data Archive must be set to the value of the running copy of the interface that was started first as defined by the **`/UFO_ID`** startup command-line parameter. • The heartbeat control points for both copies of the interface in the PI Data Archive must be changing values at a rate specified by the

Step	Description
	/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 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 PI. Backup interface not available.`" The backup interface is now considered primary and the previous primary interface is now backup. 15. Verify data in the PI Data Archive. For hot failover, there may be an overlap of data due to the queuing of data, but there must be no data loss. For warm or cold failover, short gaps in archived data are expected. 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 PI. Backup interface available.`" In the 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 log file. 18. Test failover with different failure scenarios (for example, loss of PI Data Archive connection for a single interface copy). UniInt hot failover guarantees no data loss with a single point of failure; verify no data loss by checking the data in the PI Data Archive and on the data source. For warm or cold failover, short gaps in archived data can occur. 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 to the ID 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 PI Point when the interface is stopped. This prevents the digital state being written to PI Points while a redundant system is also writing data to the same PI 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 synchronization file *path* 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 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 points as well as how often UniInt checks on the status of the other copy of the interface.	50 – 20000 / **5000**

Parameter	Required/ Optional	Description	Value/Default
`/Host=server`	Required	Host PI Data Archive for Exceptions and PI point updates The value of the `/Host` startup parameter depends on the PI Data Archive configuration. If the PI Data Archive 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 PI Data Archive 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 PI Data Archives 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.

OSIsoft 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 the PI Data Archive. **ActiveID** must be initialized so that when the interfaces read it for the first time, it is not an error. Active ID can also be used to force failover. For example, if the current primary is IF-Node 1 and Active ID is 1, you can manually change Active ID 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

PI Points

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

Active_ID Point Configuration

Attributes	**ActiveID**
Tag	**<Intf>_ActiveID**
Compmax	**0**
ExDesc	[UFO2_ActiveID]
Location1	Match # in **`/id=#`**
Location5	Optional, Time in min 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 Point 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 min to wait for backup to collect data before failing over.	Optional, Time in min to wait for backup to collect data before failing over.	Optional, Time in min to wait for backup to collect data before failing over.	Optional, Time in min 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 Point Configuration

Attribute	**Primary**	**Backup**
Tag	<Tagname1>	<Tagname2>
Compmax	0	0
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

Attribute	Primary	Backup
Shutdown	0	0
Step	1	1

The following table describes the extended descriptor for the above PI points in more detail.

PI Point ExDesc	Required / Optional	Description	Value
[UFO2_ACTIVEID]	Required	Active ID point 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 point 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' **/ps** parameter. 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 point 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' **/ps** parameter. Location1 must match the id for the interfaces.	0 – 31 / **None** Updated by the Interface on IF-Node2

PI Point ExDesc	Required / Optional	Description	Value
[UFO2_DEVICESTAT :#] (IF-Node1)	Required	Device Status 1 point The ExDesc must start with the case sensitive string: [UFO2_DEVICESTAT:#] The value following the colon (:) must be the Failover ID for the interface running on IF-Node1 The PointSource must match the interfaces' **/ps** parameter. 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 point is very similar to the UniInt Health Device Status point except the data written to this point 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 point is updated.	0 – 99 / **None** Updated by the Interface on IF-Node1
[UFO2_DEVICESTAT :#] (IF-Node2)	Required	Device Status 2 point The ExDesc must start with the case sensitive string: [UFO2_DEVICESTAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2 The PointSource must match the interfaces' **/ps** parameter. 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 point 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 point is recommended. The failover state points are digital points 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.

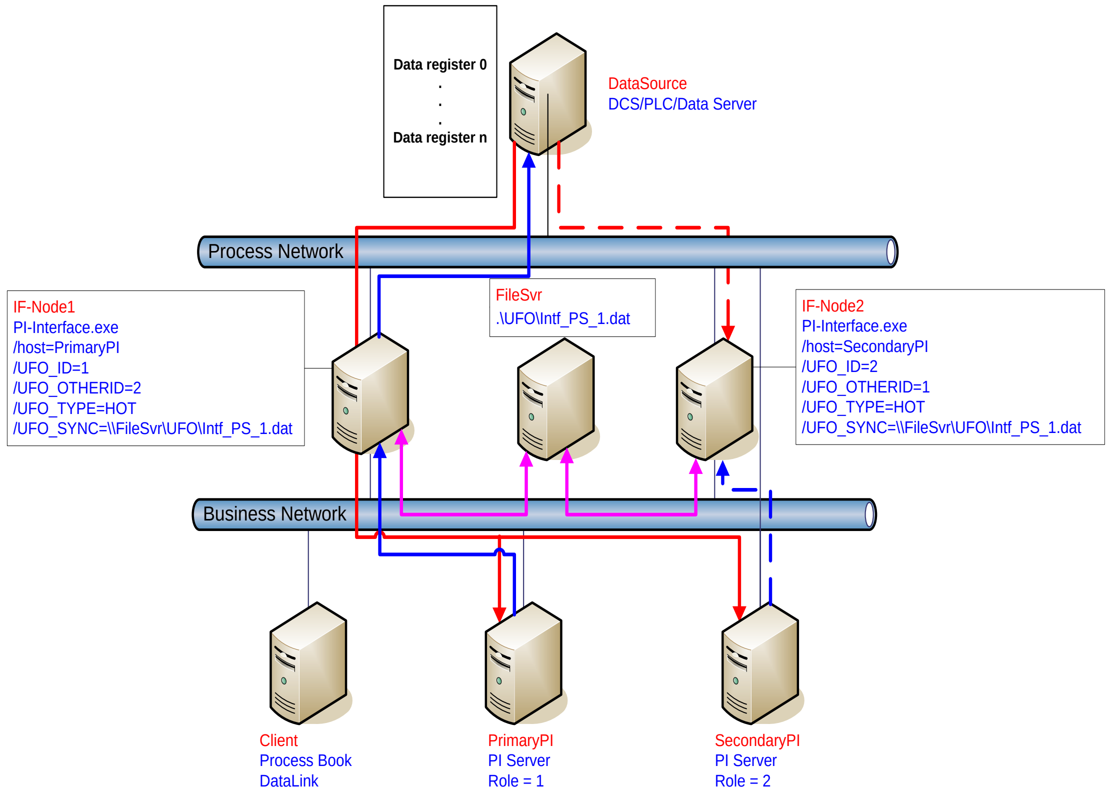
PI Point ExDesc	Required / Optional	Description	Value
		interface is running but cannot communicate with the data source. 2 = Backup No PI Connection: The interface is running and connected to the data source but has lost its communication to the PI Data Archive. 3 = Backup: The interface is running 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 the PI Data Archive.	
[UFO2_STATE:#] (IF-Node2)	Optional	State 2 point 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 point is recommended.	Normally updated by the Interface currently in the Primary state. Values range between 0 and 5. See description of State 1 point.

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

In a shared file failover configuration, no failover control information is passed between the data source and the interface. This failover scheme uses five PI points 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 PI points is not required for the proper operation of failover (unless connection to the shared file is lost). This solution does not require a connection to the PI Data Archive after initial startup because the control point data are set and monitored in the shared file. However, the PI point values are sent to the PI Data Archive so that you can monitor them with standard OSIsoft client tools.

You can force manual failover by changing the active ID point on the PI Data Archive to the backup failover ID.



The preceding figure 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. OSIsoft 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 PI Data Archive 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 PI Data Archive to determine the status of the primary interface. If the primary interface is still communicating with the PI Data Archive, than failover to the backup is not required. However, if the primary interface is not posting data to the PI Data Archive, 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 PI 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 PI 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 PI points. 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 active ID 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 the PI Data Archive. 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 PI points 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 PI points.

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 points in the PI Data Archive. Updating of the points in the PI Data Archive is assumed to take place unless communication with the PI Data Archive is interrupted. The updates to the PI Data Archive 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 PI Data Archive 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 PI Data Archive is restored, the heartbeat values will revert back to the 1 to 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

During steady state, the active ID 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 active ID 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 active ID point in the PI Data Archive to the active ID 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 PI points. 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 the PI points. This queued data is sent to the PI points using the same function calls that would have been used had the interface been in a primary state when the data was collected. If UniInt receives data without a timestamp, the primary copy uses the current PI time to timestamp data sent to PI points. Likewise, the backup copy timestamps data it receives without a timestamp with the current PI time before queuing its data. This preserves the accuracy of the timestamps.

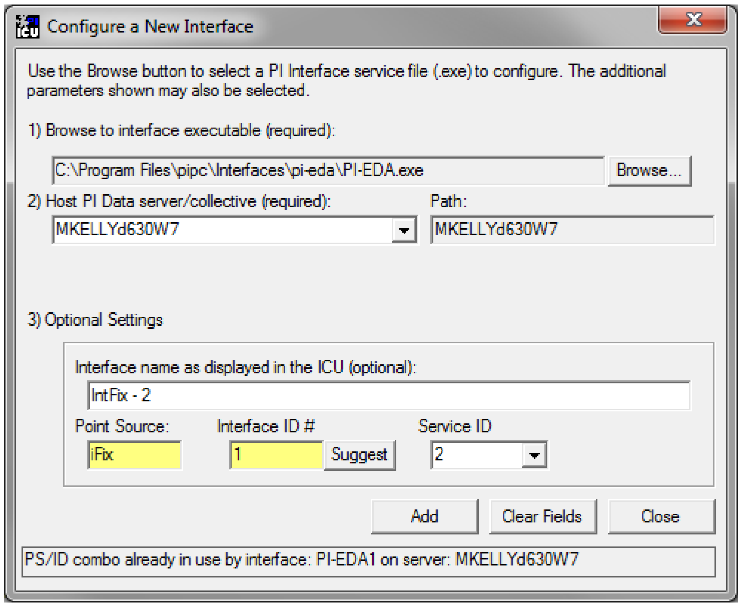
Failover Configuration Using PI ICU

The use of the PI 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 PI ICU as described in the Configuring the Interface with the PI 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 PI 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 4 below. Ignore this message and click the *Add* button.

Create the Interface Instance with PI 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\pi-eda\PI-EDA.exe

2) Host PI Data server/collective (required): MKELLYd630W7

Path: MKELLYd630W7

3) Optional Settings

Interface name as displayed in the ICU (optional): IntFix - 2

Point Source: iFix

Interface ID #: 1

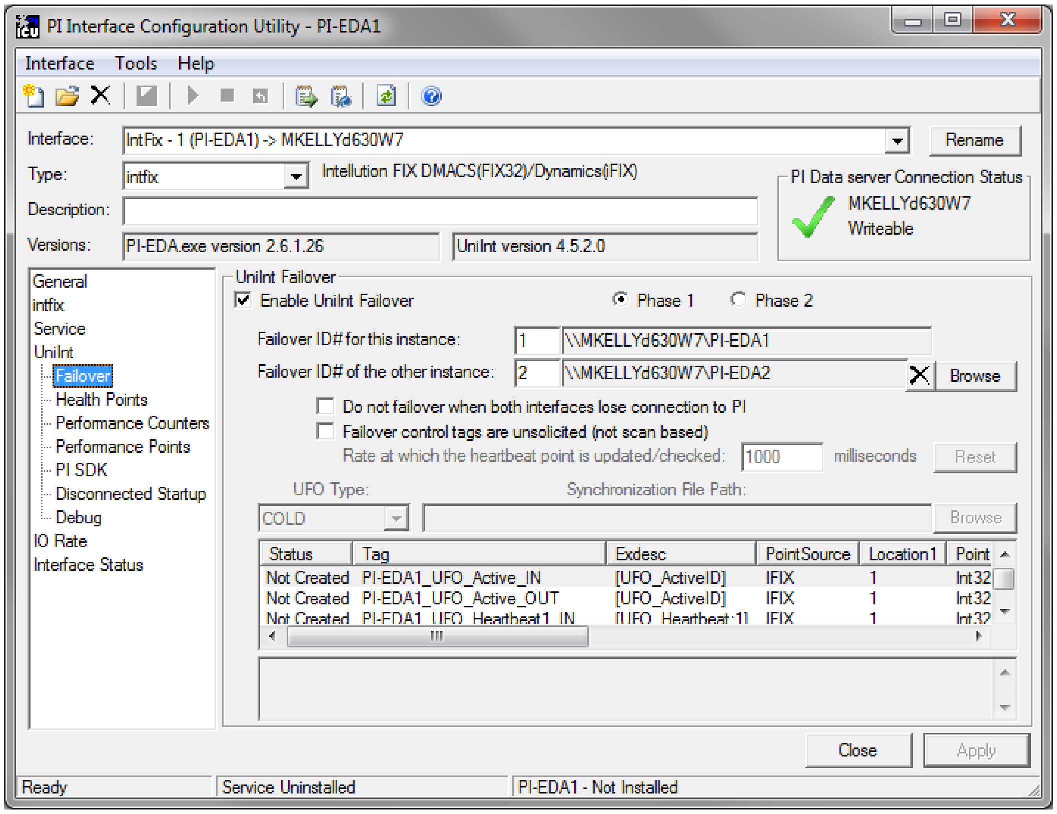
Service ID: 2

PS/ID combo already in use by interface: PI-EDA1 on server: MKELLYd630W7

Figure 4: PI 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 PI 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 1)section and Start-Up Parameters



Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7

Type: intfix Intellution FIX DMACS(FIX32)/Dynamics(iFIX)

Description:

Versions: PI-EDA.exe version 2.6.1.26 UniInt version 4.5.2.0

PI Data server Connection Status
MKELLYd630W7
Writeable

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

UniInt Failover

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

Failover ID# for this instance: 1 \\MKELLYd630W7\PI-EDA1

Failover ID# of the other instance: 2 \\MKELLYd630W7\PI-EDA2

☐ Do not failover when both interfaces lose connection to PI

☐ Failover control tags are unsolicited (not scan based)

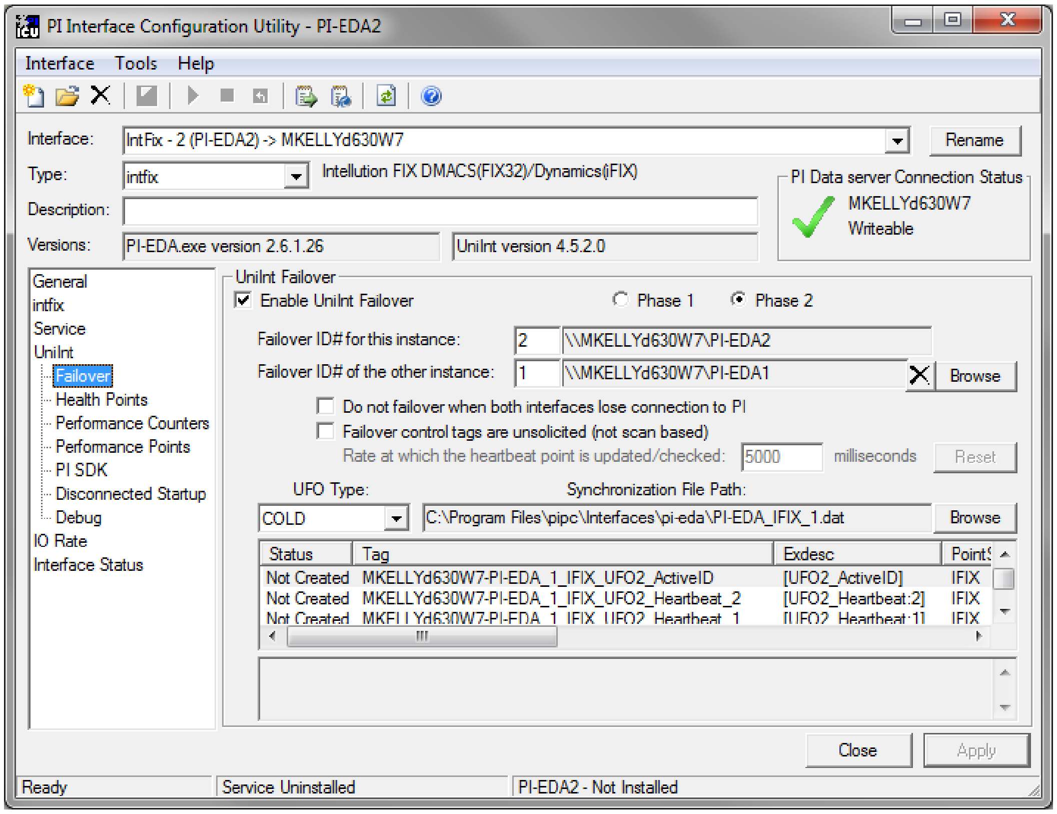
Rate at which the heartbeat point is updated/checked: 1000 milliseconds

UFO Type: COLD Synchronization File Path:

Status	Tag	Exdesc	PointSource	Location1	Point
Not Created	PI-EDA1_UFO_Active_IN	[UFO_ActiveID]	IFIX	1	Int32
Not Created	PI-EDA1_UFO_Active_OUT	[UFO_ActiveID]	IFIX	1	Int32
Not Created	PI-EDA1_UFO_Heartbeat1_IN	[UFO_Heartbeat:1]	IFIX	1	Int32

Ready | Service Uninstalled | PI-EDA1 - Not Installed

Figure 5: This figure illustrates the PI ICU failover configuration page showing the UniInt failover startup parameters (phase 1). This copy of the interface defines its failover ID as 1 (**`/UFO_ID=1`**) and the other interface's failover ID as 2 (**`/UFO_OtherID=2`**). The other failover interface copy must define its failover ID as 2 (**`/UFO_ID=2`**) and the other interface failover ID as 1 (**`/UFO_OtherID=1`**) in its ICU failover configuration page.



Interface: IntFix - 2 (PI-EDA2) -> MKELLYd630W7
Type: intfix Intellution FIX DMACS(FIX32)/Dynamics(iFIX)
Description:
Versions: PI-EDA.exe version 2.6.1.26 UniInt version 4.5.2.0
PI Data server Connection Status: MKELLYd630W7 Writeable

UniInt Failover
[✓] Enable UniInt Failover Phase 1 [✓ Phase 2]
Failover ID# for this instance: 2 \\MKELLYd630W7\PI-EDA2
Failover ID# of the other instance: 1 \\MKELLYd630W7\PI-EDA1
[ ] Do not failover when both interfaces lose connection to PI
[ ] Failover control tags are unsolicited (not scan based)
Rate at which the heartbeat point is updated/checked: 5000 milliseconds
UFO Type: COLD
Synchronization File Path: C:\Program Files\pipc\Interfaces\pi-eda\PI-EDA_IFIX_1.dat

Status	Tag	Exdesc	Point
Not Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_ActiveID	[UFO2_ActiveID]	IFIX
Not Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_Heartbeat_2	[UFO2_Heartbeat:2]	IFIX
Not Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_Heartbeat_1	[UFO2_Heartbeat:1]	IFIX

Ready | Service Uninstalled | PI-EDA2 - Not Installed

Figure 6: This figure illustrates the PI ICU failover configuration page 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 interface's 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 page. It also defines the location and name of the synchronization file as well as the type of failover as COLD.

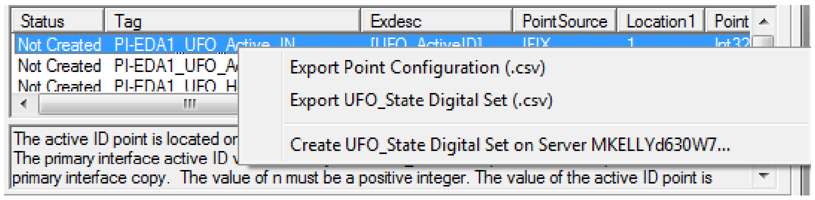
Creating the Failover State Digital State Set

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

Using the PI 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 points in the list and then click the *Create UFO_State Digital Set on Server XXXXXX…* command, where *XXXXXX* is the PI Data Archive where the points will be or are created.

Phase 1

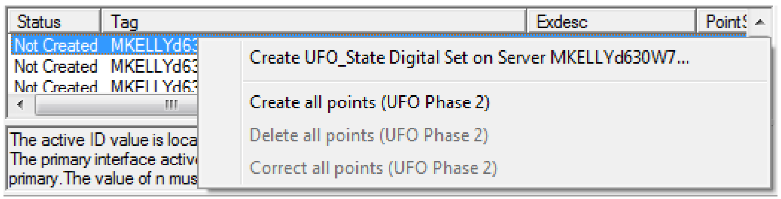


Status	Tag	Exdesc	PointSource	Location1	Point
Not Created	PI-EDA1_UFO_Active_IN	[UFO_ActiveID]	IFIX	1	Int32
Not Created	PI-EDA1_UFO_A				
Not Created	PI-EDA1_UFO_H				

The active ID point is located or
The primary interface active ID v
primary interface copy. The value of n must be a positive integer. The value of the active ID point is

Export Point Configuration (.csv)
Export UFO_State Digital Set (.csv)
Create UFO_State Digital Set on Server MKELLYd630W7...

Phase 2



Status	Tag	Exdesc	Point:
Not Created	MKELLYd63		
Not Created	MKELLYd63		
Not Created	MKELLYd63		

The active ID value is loca
The primary interface activ
primary. The value of n mus

Create UFO_State Digital Set on Server MKELLYd630W7...
Create all points (UFO Phase 2)
Delete all points (UFO Phase 2)
Correct all points (UFO Phase 2)

This command is unavailable if the UFO_State digital state set already exists on the *XXXXXX* PI Data Archive.

Using the PI SMT 3 Utility to create Digital State Set

Optionally the *Export UFO_State Digital Set (.csv)* command on the shortcut menu can be selected to create a comma-separated file to be imported via the PI System Management Tools (SMT) (version 3.0.0.7 or later) 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 in a PI Data Archive using the *Import from File* command found in the SMT application. The procedure assumes the user has a basic understanding of the SMT application.

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

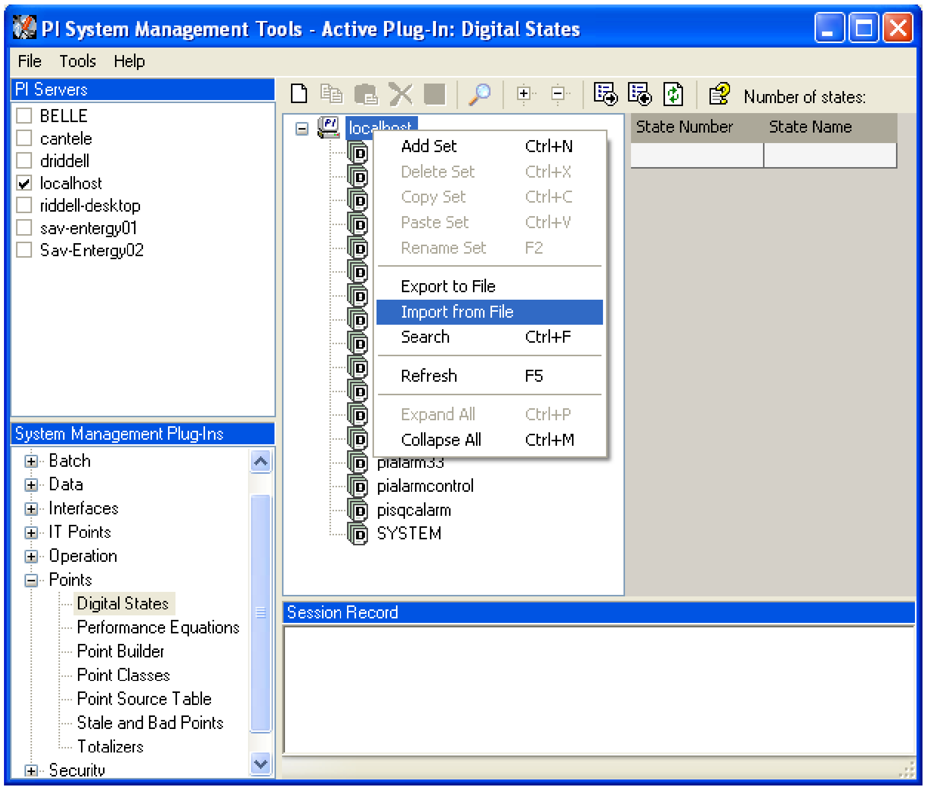
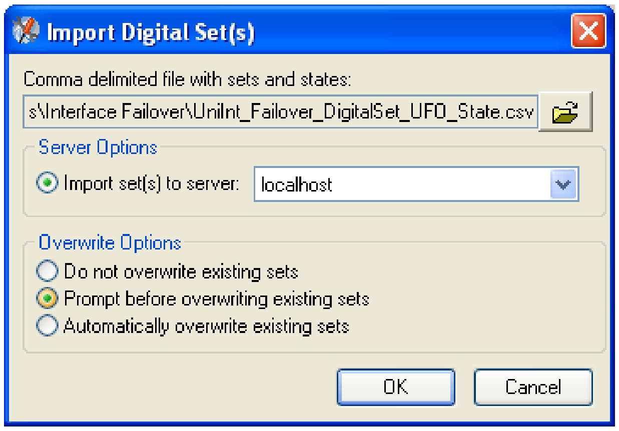


Figure 7: PI SMT application configured to import a digital state set file. The *PI Servers* window shows the “localhost” PI Data Archive selected along with the System Management Plug-Ins window showing the Digital States Plug-In as being selected. The digital state set file can now be imported by selecting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

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*. Refer to Figure 8 below.



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

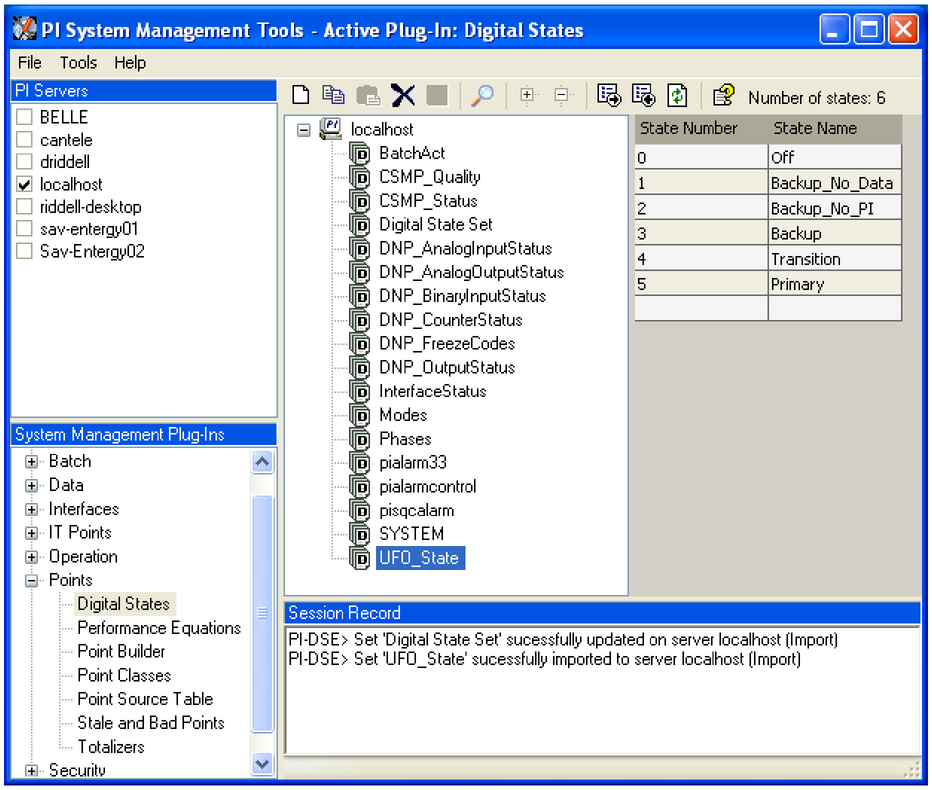
Server Options
[✓ Import set(s) to server:] localhost

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

Figure 8: PI SMT application *Import Digital Set(s)* dialog box. 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 option button.

6. Click on the *OK* button. Refer to Figure 8 above.

7. 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 PI SMT application showing the `UFO_State` digital set created in the "localhost" PI Data Archive.

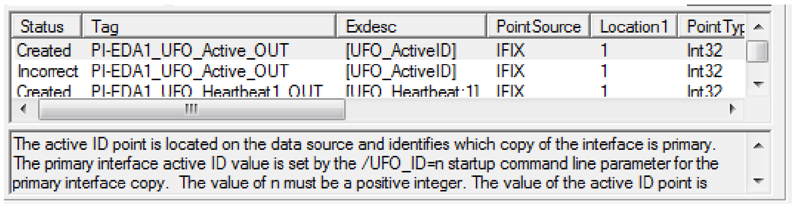
Creating the UniInt Failover Control and Failover State Points (Phase 1)

The ICU can be used to create a comma delimited file that contains all of the non-interface specific point attributes configured correctly for UniInt failover. This file can be edited according to the UniInt failover point configuration sections above.

In addition, the interface installation procedure installs an example file that creates the UniInt failover points. This example file already has the IntFix Interface specific attributes configured.

To use the ICU *Failover* page to create this file, simply right-click any of the failover points in the list and select *Export Point Configuration* then edit this file as needed and import with SMT.

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



Status	Tag	Exdesc	PointSource	Location1	PointTyp
Created	PI-EDA1_UFO_Active_OUT	[UFO_ActiveID]	IFIX	1	Int32
Incorrect	PI-EDA1_UFO_Active_OUT	[UFO_ActiveID]	IFIX	1	Int32
Created	PI-EDA1_UFO_Heartbeat1_OUT	[UFO_Heartbeat:1]	IFIX	1	Int32

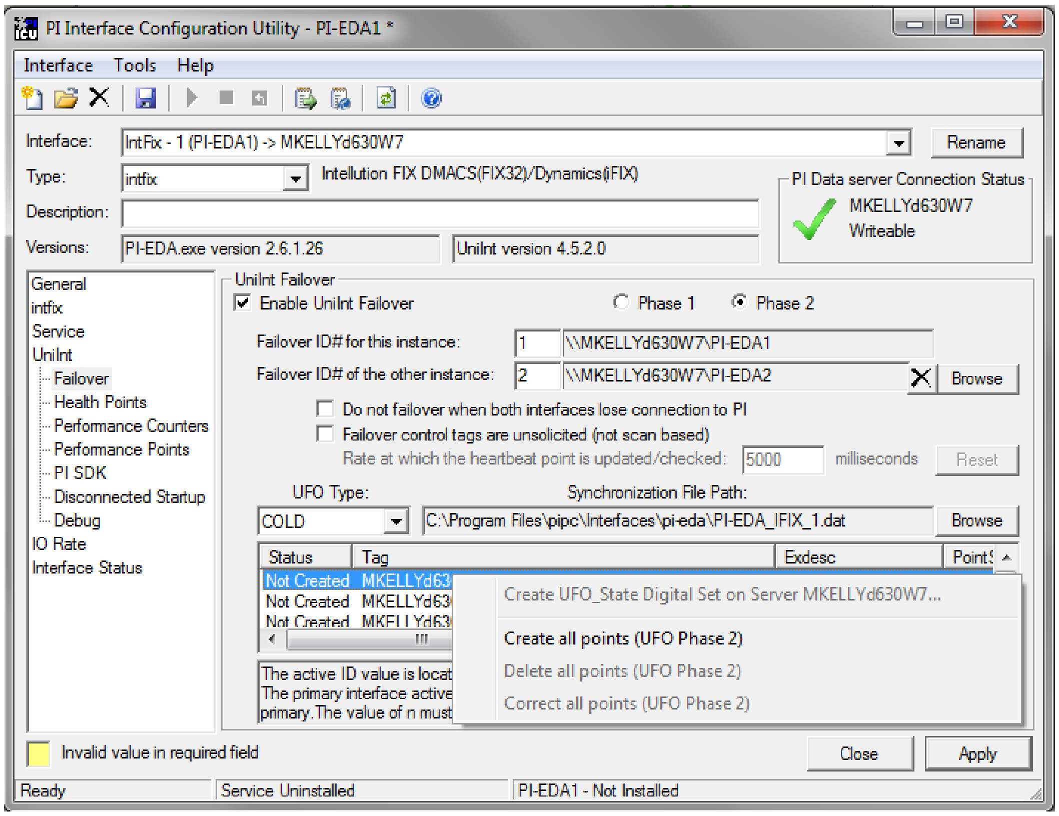
The active ID point is located on the data source and identifies which copy of the interface is primary. The primary interface active ID value is set by the /UFO_ID=n startup command line parameter for the primary interface copy. The value of n must be a positive integer. The value of the active ID point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

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

The ICU can be used to create the UniInt failover control and state points.

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

If this menu command is unavailable, it is because the UFO_State digital state set has not been created in the PI Data Archive yet. *Create UFO_State Digital Set on Server xxxxxxx…* on the shortcut menu can be used to create that digital state set. After this has been done, then the *Create all points (UFO Phase2)* command should be available.



PI Interface Configuration Utility - PI-EDA1 *
Interface Tools Help
Rename
Browse
Reset
Close
Apply
General
intfix
Service
UniInt
Failover
Health Points
Performance Counters
Performance Points
PI SDK
Disconnected Startup
Debug
IO Rate
Interface Status
Create UFO_State Digital Set on Server MKELLYd630W7...
Create all points (UFO Phase 2)
Delete all points (UFO Phase 2)
Correct all points (UFO Phase 2)

Interface: IntFix - 1 (PI-EDA1) -> MKELLYd630W7

Type: intfix Intellution FIX DMACS(FIX32)/Dynamics(iFIX)

Description:

Versions: PI-EDA.exe version 2.6.1.26 UniInt version 4.5.2.0

PI Data server Connection Status: MKELLYd630W7 Writeable

UniInt Failover

[✓] Enable UniInt Failover ( ) Phase 1 (●) Phase 2

Failover ID# for this instance: 1 \\MKELLYd630W7\PI-EDA1

Failover ID# of the other instance: 2 \\MKELLYd630W7\PI-EDA2

[ ] Do not failover when both interfaces lose connection to PI

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

Rate at which the heartbeat point is updated/checked: 5000 milliseconds

UFO Type: COLD

Synchronization File Path: C:\Program Files\pipc\Interfaces\pi-eda\PI-EDA_IFIX_1.dat

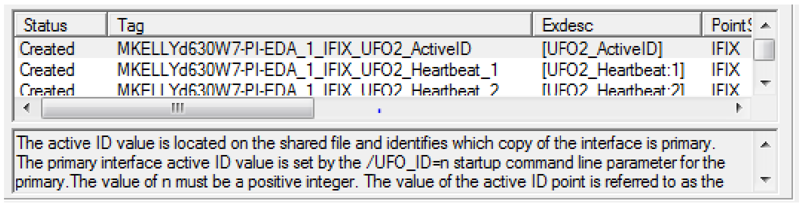
Status	Tag	Exdesc	Point
Not Created	MKELLYd63		
Not Created	MKELLYd63		
Not Created	MKELLYd63		

The active ID value is locat
The primary interface active
primary.The value of n must

Invalid value in required field

Ready | Service Uninstalled | PI-EDA1 - Not Installed

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



Status	Tag	Exdesc	Point
Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_ActiveID	[UFO2_ActiveID]	IFIX
Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_Heartbeat_1	[UFO2_Heartbeat:1]	IFIX
Created	MKELLYd630W7-PI-EDA_1_IFIX_UFO2_Heartbeat_2	[UFO2_Heartbeat:2]	IFIX

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

Converting from Phase 1 to Phase 2 Failover

The few differences between Phase 1 and Phase 2 Failover are described in the following table.

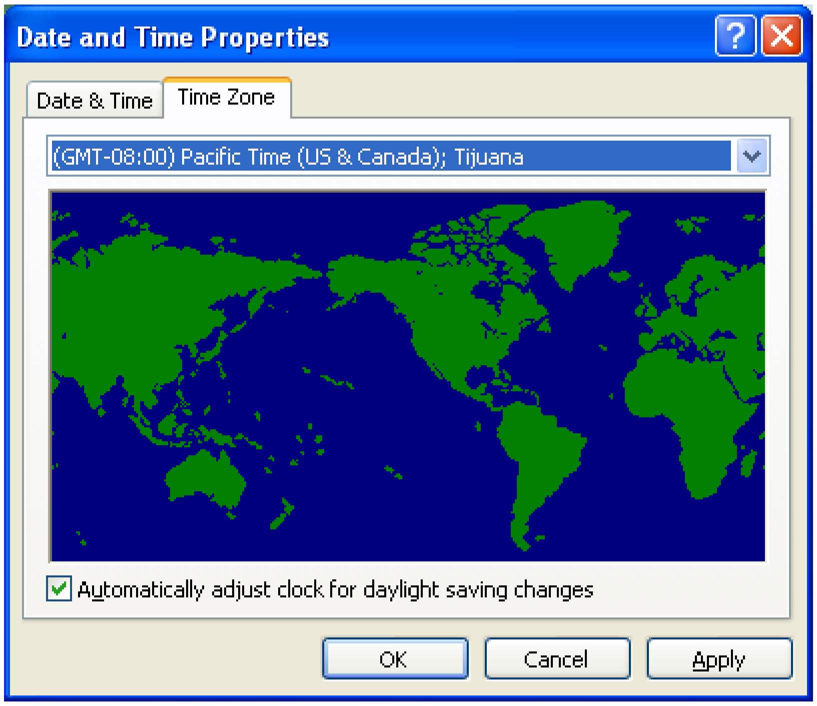
Points / Attributes / Parameters	**Phase 1**	**Phase 2**
Control Data Path parameter	Absence of this command line parameter (**`/UFO_Sync=<path>`**) in the startup (.bat) file, causes Phase 1 synchronization.	The presence of this command line parameter (**`/UFO_Sync=<path>`**) signals Phase 2 synchronization and specifies the directory path to the shared file and, optionally, the file name.
Control Points	Six PI pointss	Five PI points
	Phase 2 does not require both input and output points because they are not serviced by the interface client. Phase 2 failover requires only a single Active ID, Heartbeat 1, and Heartbeat 2 point.	
	Active ID (input)	Active ID
	Active ID (output)	
	Heartbeat 1 (input)	Heartbeat 1
	Heartbeat 1 (output)	
	Heartbeat 2 (input)	Heartbeat 2
	Heartbeat 2 (output)	
	Phase 2 requires two device status points to convey the status of communications link to the data source.	
		DeviceStatus 1
		DeviceStatus 2
	[Optional] State points	[Optional] State points
InstrumentTag attributes	Phase 1 requires these point attributes to communicate directly with the data source device.	
ExDesc attributes	Keyword [UFO_tagname]	Keyword [UFO2_tagname]

Procedure

Step	**Description**
1.	Add **`/UFO_Sync`** parameter to the startup file to define the path and, optionally, the file name of the shared synchronization file.
2.	Change the Active ID, Heartbeat 1, and Heartbeat 2 points to remove input/output designations OR Create new Phase 2 points that do not have input/output qualifiers.
3.	Create device status 1 and device status 2 points.
4.	Create new points for Phase 2 or change the ExDesc attribute keywords for the points from [**UFO**_tagname] to [**UFO2**_tagname].
5.	Remove InstrumentTag attributes.

Chapter 11. 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 12. Security

The PI Firewall Database and the PI Trust Database must be configured so that the interface is allowed to write data to the PI Data Archive.

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

See “Manage Interface Authentication with PI Trusts” in the chapter “Manage Security” of the *PI Server Introduction to System Management Guide*.

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

Authentication

Interface instances are usually configured to run as Windows services. Since a service runs in a non-interactive context, a PI trust is required to authenticate the interface service to the PI Data Archive. A PI trust is associated with one PI identity, PI user, or PI group. When an interface successfully authenticates through a trust, the interface is granted the access rights for the associated identity, user, or group.

OSIsoft discourages using highly-privileged identities, users, or groups in PI trusts for interfaces.

Security Note: Avoid using the piadmin super-user and piadmins group. The recommended best practice for PI Data Archive security is to create an identity, user, or group that has only the access rights which are necessary for the interface to operate.

PI Data Archive v3.3 and Later

Security Configuration using Trust Editor

The Trust Editor plug-in for PI System Management Tools edits the PI trust table.

See the “Manage Interface Authentication with PI Trusts” section in the *PI Server Introduction to System Management* manual for more details on security configuration.

Security configuration using piconfig

For PI Data Archive v3.3 and higher, the following example demonstrates how to edit the PI trust table with piconfig:

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

For the preceding example,

`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 identity, user, or group the interface is entrusted as; in the example,

`trust_identity`

PI Data Archive v3.2

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

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

In place of *`piapimachine`*, put the name of the interface node *as it is seen by the PI Data Archive*.

Authorization

For an interface instance to start and write data to PI points, the following permissions must be granted to the PI identity, user, or group in the PI trust that authenticates the interface instance.

Database Security	Permission	Notes
PIPOINT	r,w	

Point Database	Permission	Notes
PtSecurity	r,w	
DataSecurity	r,w	Unbuffered
	r	Buffered (the buffering application requires r,w for the interface points)

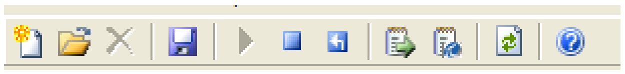
The permissions in the preceding table must be granted for every PI point that is configured for the interface instance. Observe that buffering on the interface node is significant to PI point permissions.

When the interface instance is running on an unbuffered interface node, the interface instance sends PI point updates directly to the PI Data Archive. Therefore, the DataSecurity attribute must grant write access to the PI identity, user, or group in the PI trust that authenticates the *interface instance*.

When the interface instance is running on a buffered interface node, the interface instance sends PI point updates to the local buffering application, which relays the PI point updates to the PI Data Archive. The buffering application is a separate client to the PI Data Archive and, therefore, authenticates independently of the interface instances. The DataSecurity attribute must grant write access to the PI identity, user, or group in the PI trust that authenticates the *buffering application*.

Chapter 13. Starting / Stopping the Interface

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



Starting Interface as a Service

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

```
PI-EDA.exe /start [ /serviceid id ]
```

To start the interface service with PI ICU, use the ▶ button on the PI 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 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 PI ICU, the Services control panel or with the command:

```
PI-EDA.exe /stop [ /serviceid id ]
```

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

The service can be removed by:

```
PI-EDA.exe /remove [ /serviceid id ]
```

Chapter 14. 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 PI Data Archives. OSIsoft strongly recommends that you enable buffering on your interface nodes. Otherwise, if the interface node stops communicating with the PI Data Archive, 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 PI Data Archives that are part of a 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 PI Data Archives in a collective. (Bufserv also supports n-way buffering, but OSIsoft 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 PI Data Archive collective, PIBufss guarantees identical data in the archive records of all the PI Data Archives that are part of that collective.

You can use PIBufss only under the following conditions:

- the PI Data Archive version is at least 3.4.375.x; and
- all of the interfaces running on the interface node send data to the same PI Data Archive or to the same collective.

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

- the PI Data Archive version is earlier than 3.4.375.x; or
- the interface node runs multiple interfaces, and these interfaces send data to multiple PI Data Archives that are not part of a single collective.

If an interface node runs multiple interfaces, and these interfaces send data to two or more PI Data Archive 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 collective, you need to use two or more interface nodes to run your interfaces.

It is technically possible to run Bufserv on the PI Data Archive node. However, OSIsoft 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 PI Data Archive. It also creates shared memory storage.

When an interface program makes a PI API function call that writes data to the PI Data Archive (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 PI Data Archive. 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 PI Data Archive exists, sends the data to the PI Data Archive; or
- if there is no connection to the PI Data Archive, 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 PI Data Archive, it writes to the PI Data Archive the interface data contained in both shared memory storage and disk.

(Before sending data to the PI Data Archive, PIBufss performs further tasks such as 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 configuration where 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 PI Data Archive Security

After you enable buffering, it is the buffering application - and not the interface program - that writes data to the PI Data Archive. If the PI Data Archive's trust table contains a trust entry that allows all applications on an interface node to write data, then the buffering application is able to write data to the PI Data Archive.

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

Buffering Application	Application Name field for PI Trust
PI 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)

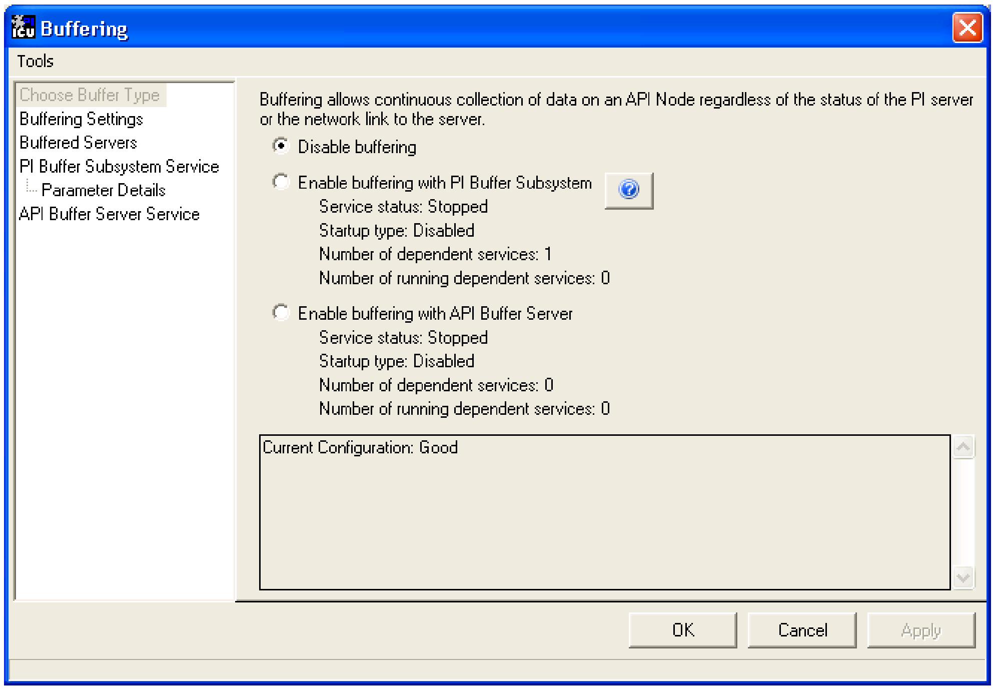
To use a process name greater than 4 characters in length for a trust application name, use the LONGAPPNAME=1 in the PIClient.ini file.

See the Security chapter for additional information.

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

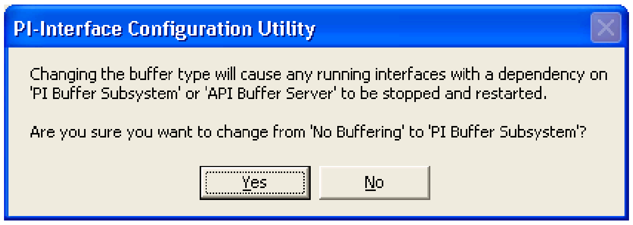
( ) Enable buffering with API Buffer Server
Service status: Stopped
Startup type: Disabled
Number of dependent services: 0
Number of running dependent services: 0

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.

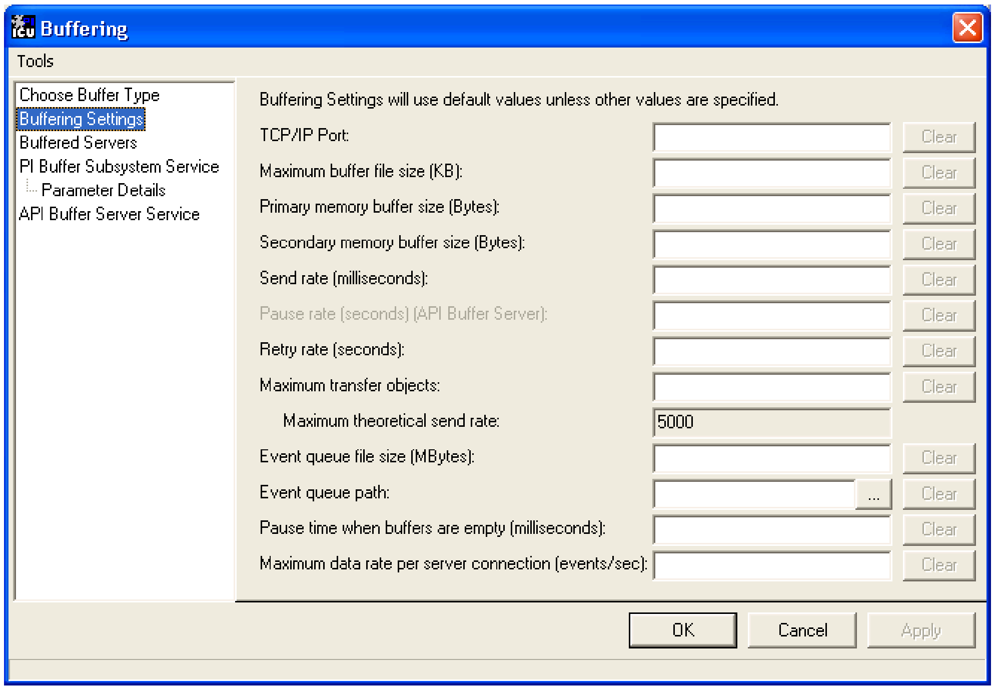
Are you sure you want to change from 'No Buffering' to 'PI Buffer Subsystem'?

Buffering Settings

There are a number of settings that affect the operation of PIBufss and Bufserv. The *Buffering Settings* section 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 OSIsoft 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. OSIsoft 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 PI Data Archive. 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 MB (up to a maximum of 128 GB). Please see the section entitled "Queue File Sizing" in the *PIBufss.chm* file for details on how to appropriately size the event queue files.

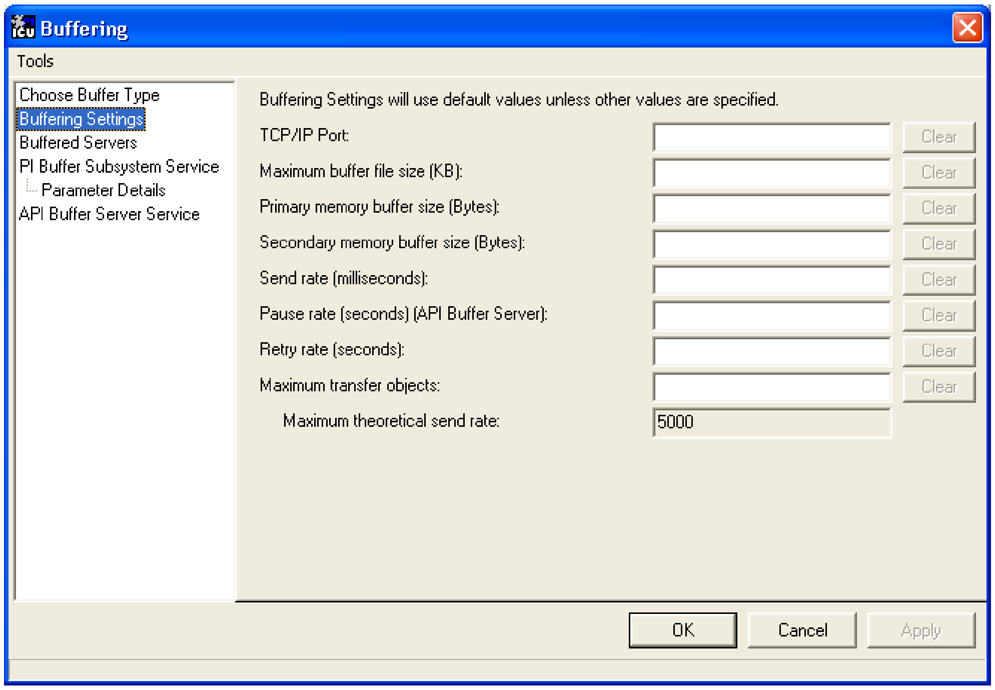
Event Queue Path

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

For optimal performance and reliability, OSIsoft 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 OSIsoft 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 PI Data Archive, 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. OSIsoft 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 PI Data Archive. The default value is 100. The valid range is 0 to 2,000,000.

Maximum transfer objects

Max 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.

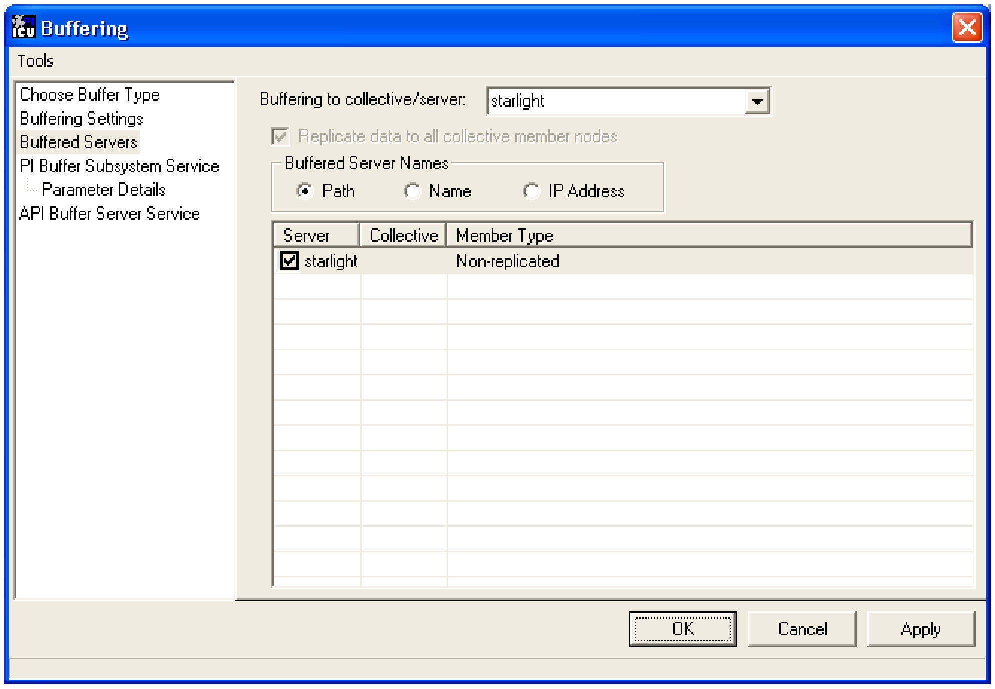
Buffered Servers

The *Buffered Servers* section allows you to define the PI Data Archives or PI Data Archive collective that the buffering application writes data.

PIBufss

PIBufss buffers data only to a single PI Data Archive or a single PI Data Archive collective. Select the PI Data Archive or the collective from the *Buffering to collective/server* drop down list box.

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



Buffering to collective/server: starlight

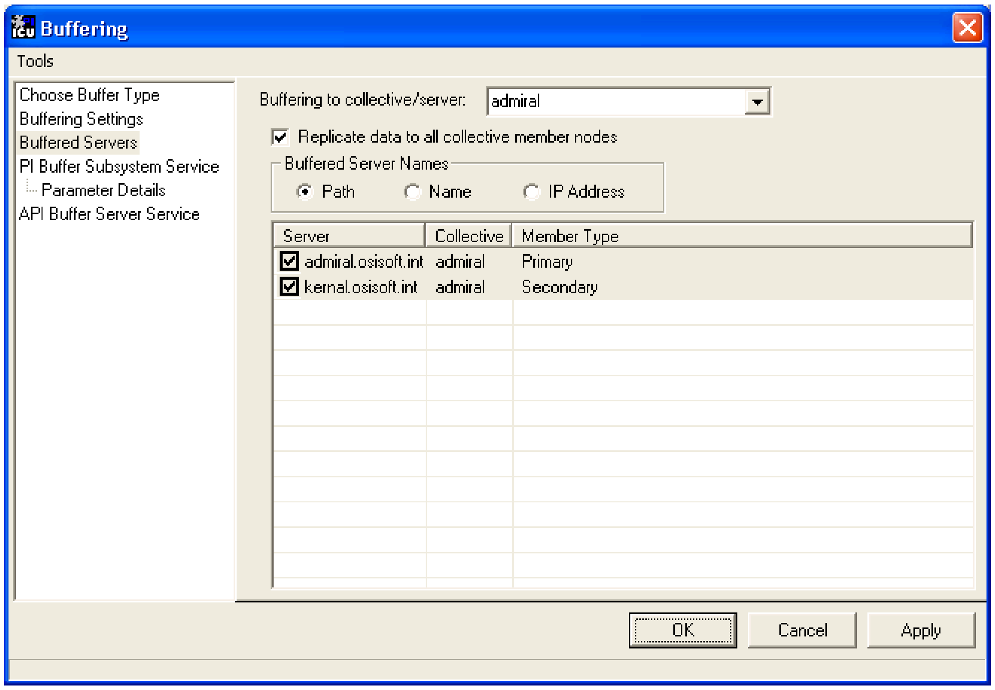
Replicate data to all collective member nodes

Buffered Server Names: [✓ Path] Name IP Address

Server	Collective	Member Type
✓ starlight		Non-replicated

The following figure shows that PIBufss is configured to write data to a PI Data Archive 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 clearing the *Replicate data to all collective member nodes* check box. Then, select (or clear) the collective members as desired.



Buffering to collective/server: admiral

✓ Replicate data to all collective member nodes

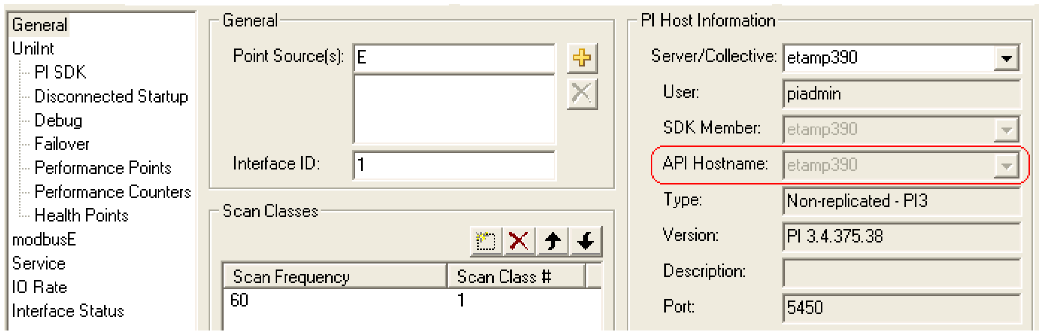
Buffered Server Names: [✓ Path] Name IP Address

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

Bufserv

Bufserv buffers data to a standalone PI Data Archive or to multiple standalone PI Data Archives. (If you want to buffer to multiple PI Data Archives that are part of a collective, you should use PIBufss.)

If the PI Data Archive 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 PI Data Archive name must be identical to the *API Hostname* entry:



General

Point Source(s): E

Interface ID: 1

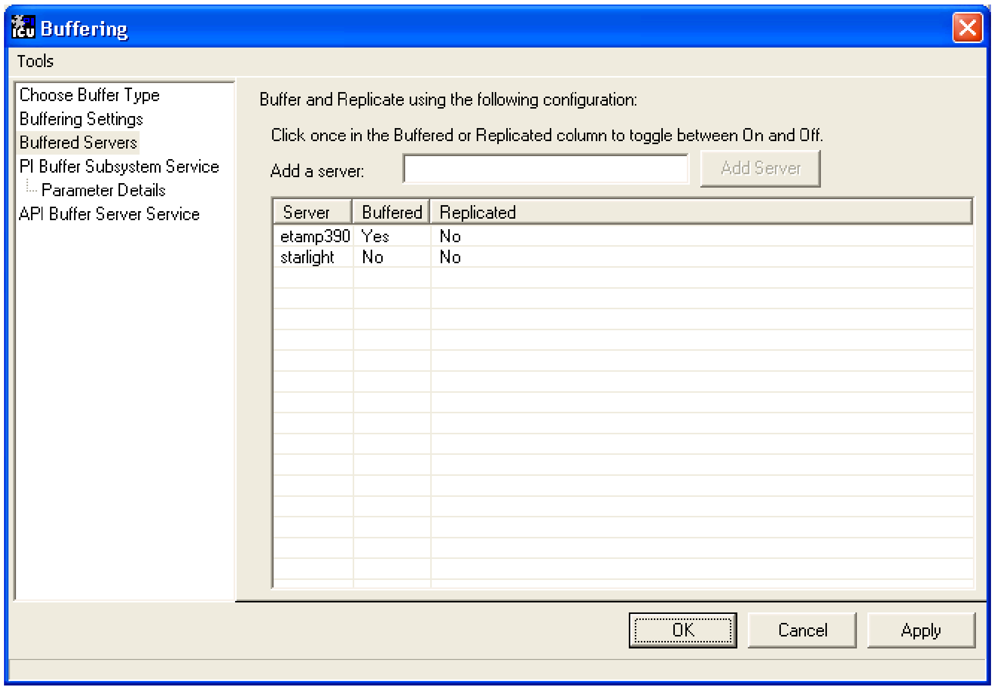
Scan Classes

Scan Frequency	Scan Class #
60	1

PI Host Information

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 named `etamp390`. You use this configuration when all the interfaces on the interface node write data to `etamp390`.



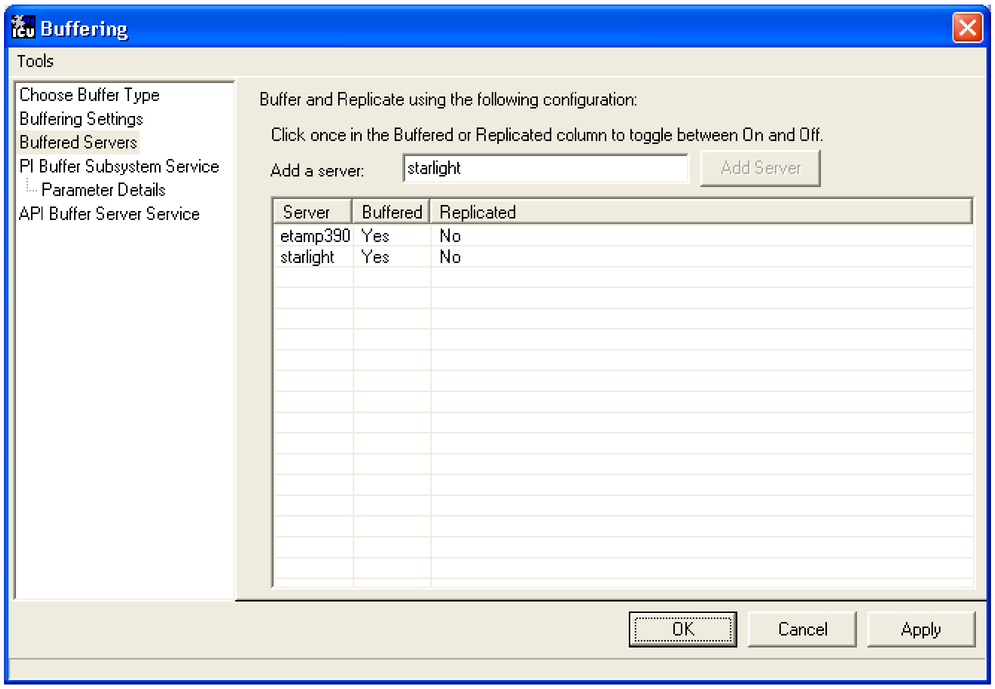
Buffer and Replicate using the following configuration:

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 PI Data Archives, 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

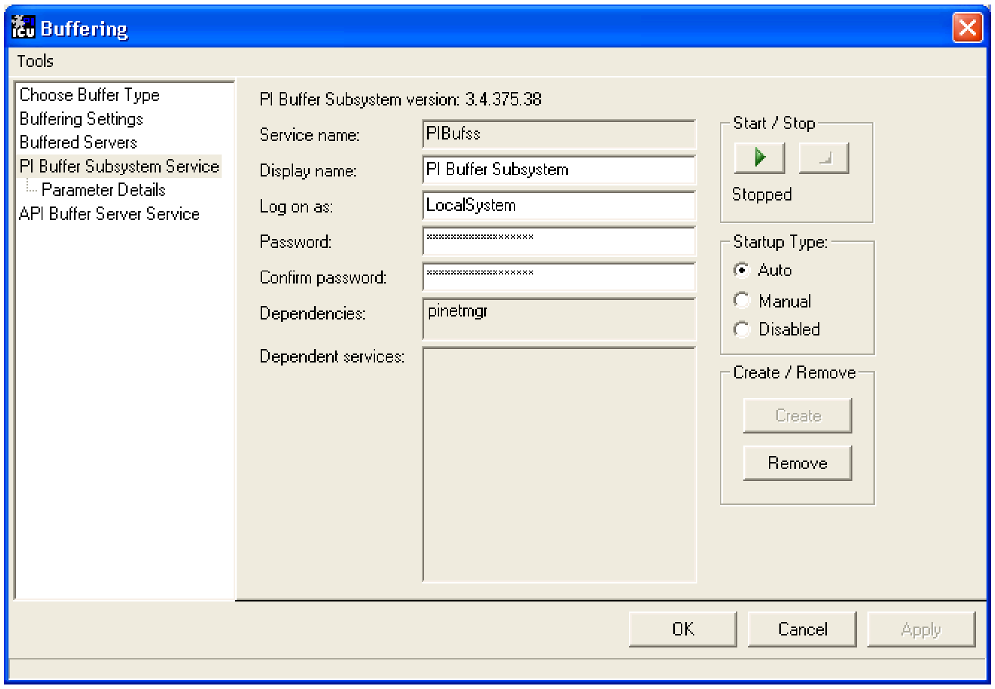
Installing Buffering as a Service

Both the PIBufss and Bufserv applications run as a Windows service.

PI Buffer Subsystem Service

Use the *PI 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.

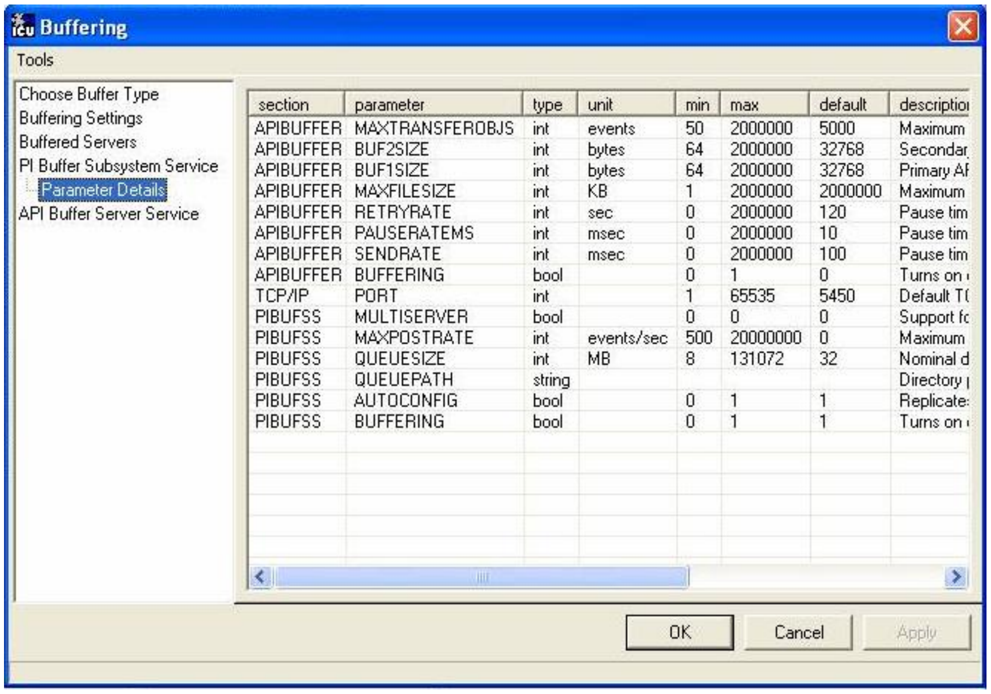


PI Buffer Subsystem version: 3.4.375.38

Service name:	PIBufss
Display name:	PI Buffer Subsystem
Log on as:	LocalSystem
Password:	******************
Confirm password:	******************
Dependencies:	pinetmgr
Dependent services:	

Stopped

Startup Type: [✓ Auto] Manual Disabled

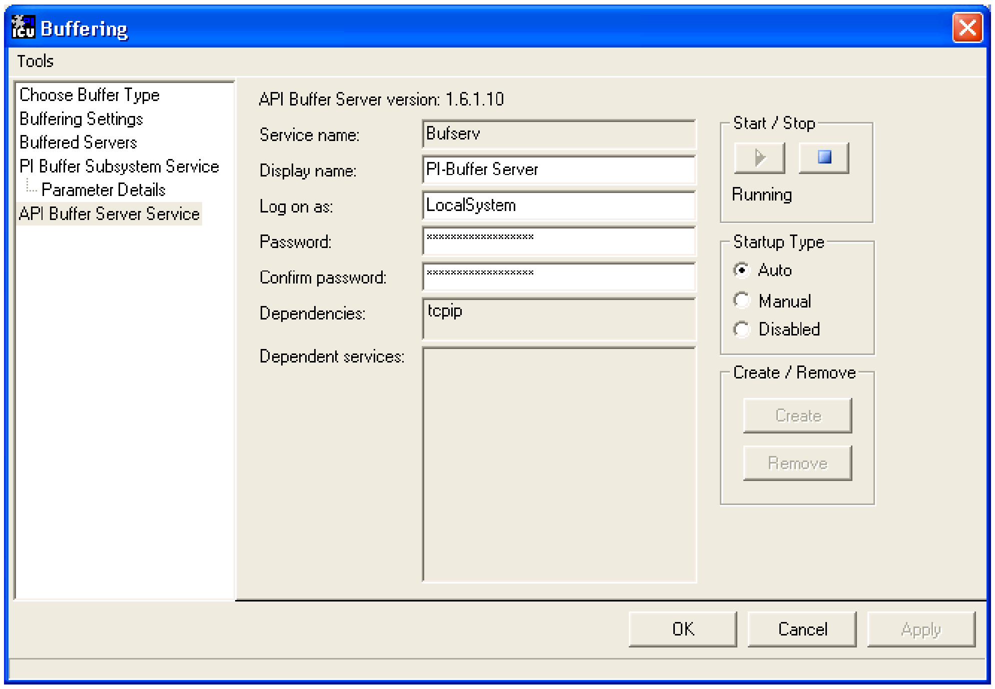


section	parameter	type	unit	min	max	default	descriptio
APIBUFFER	MAXTRANSFEROBJS	int	events	50	2000000	5000	Maximum
APIBUFFER	BUF2SIZE	int	bytes	64	2000000	32768	Secondar
APIBUFFER	BUF1SIZE	int	bytes	64	2000000	32768	Primary AF
APIBUFFER	MAXFILESIZE	int	KB	1	2000000	2000000	Maximum
APIBUFFER	RETRYRATE	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:	

Start / Stop: Running

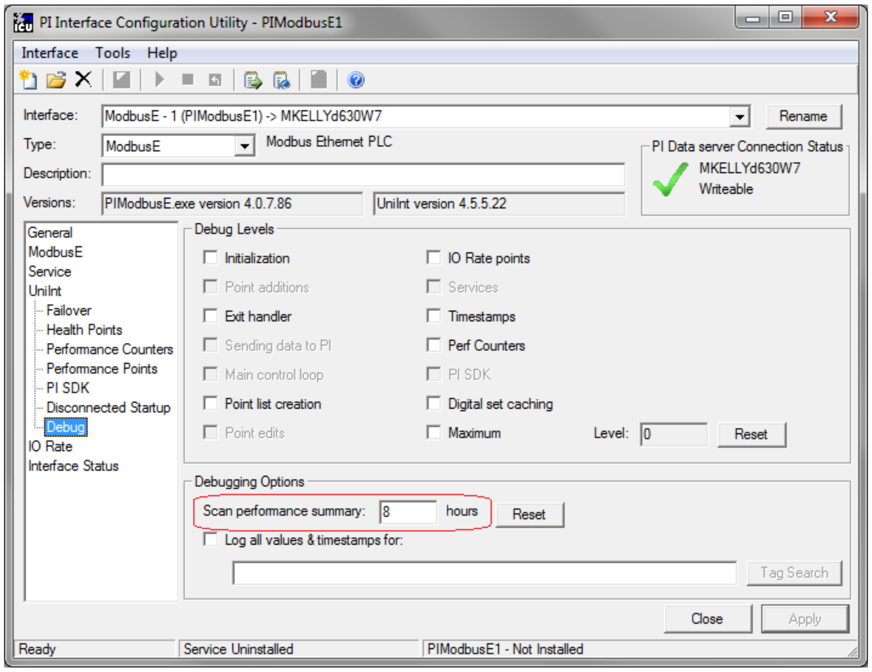
Startup Type: [✓ Auto] Manual Disabled

Chapter 15. Interface Diagnostics Configuration

The PI Point Configuration chapter provides information on building PI 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 pane:


PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 | UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Debug Levels: Initialization, Point additions, Exit handler, Sending data to PI, Main control loop, Point list creation, Point edits, IO Rate points, Services, Timestamps, Perf Counters, PI SDK, Digital set caching, Maximum — Level: 0

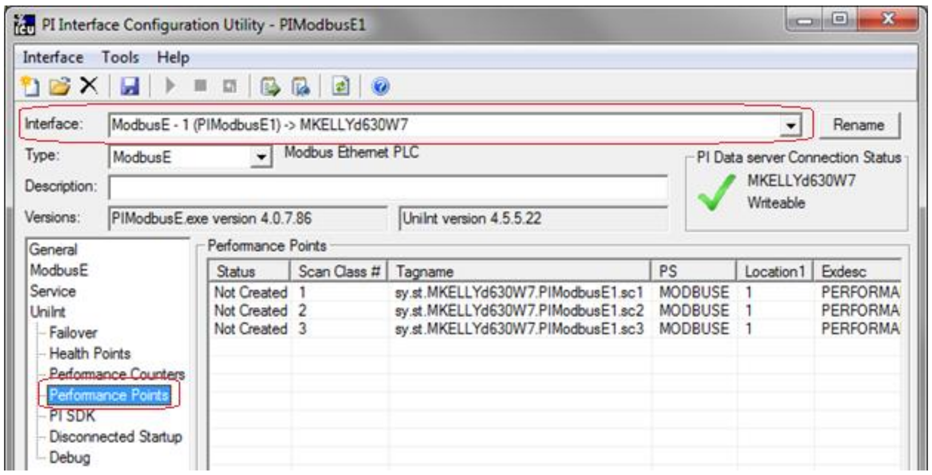
Debugging Options: Scan performance summary: 8 hours; Log all values & timestamps for:

Ready | Service Uninstalled | PIModbusE1 - Not Installed

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 - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

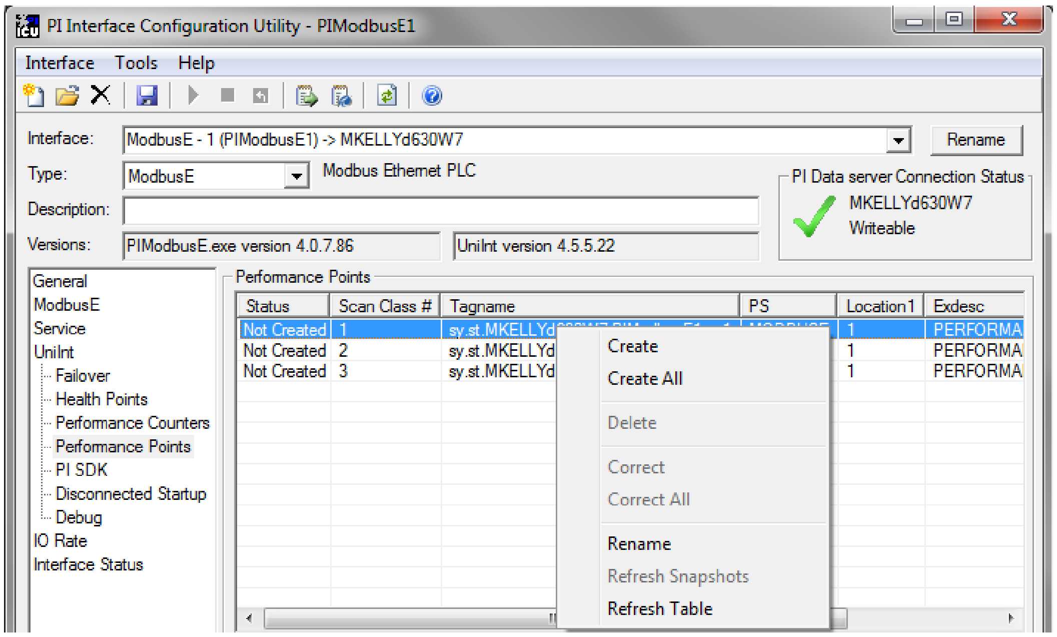
Versions: PIModbusE.exe version 4.0.7.86 — UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Performance Points

Status	Scan Class #	Tagname	PS	Location1	Exdesc
Not Created	1	sy.st.MKELLYd630W7.PIModbusE1.sc1	MODBUSE	1	PERFORMA
Not Created	2	sy.st.MKELLYd630W7.PIModbusE1.sc2	MODBUSE	1	PERFORMA
Not Created	3	sy.st.MKELLYd630W7.PIModbusE1.sc3	MODBUSE	1	PERFORMA

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



PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 — UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Performance Points

Status	Scan Class #	Tagname	PS	Location1	Exdesc
Not Created	1	sy.st.MKELLYd		1	PERFORMA
Not Created	2	sy.st.MKELLYd		1	PERFORMA
Not Created	3	sy.st.MKELLYd		1	PERFORMA

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 point 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 point 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 point to be corrected, and selecting *Correct*. The Performance Points are created with the following PI 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 PI attribute values:

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

Rename

Right-click the line belonging to the point 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* combo box on the *UniInt Parameters* tab.

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 the PI Data Archive. 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

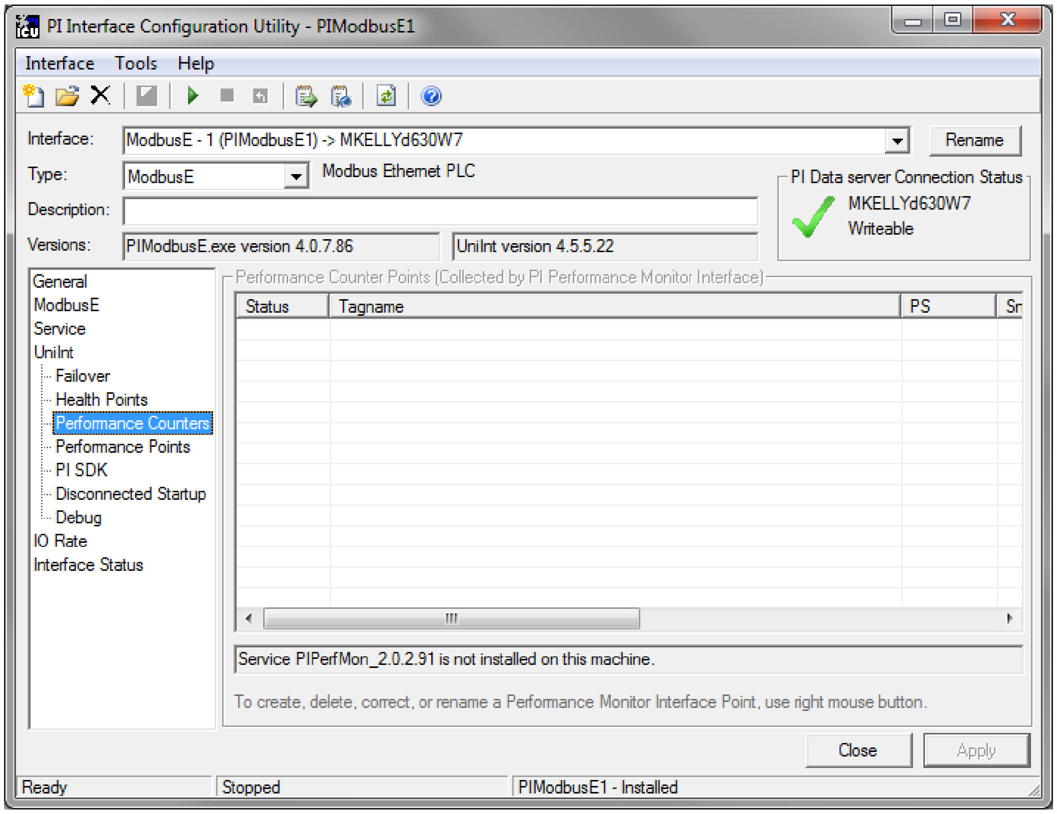
When running as a Service or interactively, this Interface exposes performance data via Windows Performance Counters. Such data include items like:

- 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 points that are currently updating among others

There are two types or instances of Performance Counters that can be collected and stored in PI 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.

OSIsoft's PI Performance Monitor Interface is capable of reading these performance values and writing them to PI points. Please see the *Performance Monitor Interface* for more information.

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


PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 | UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Performance Counter Points (Collected by PI Performance Monitor Interface)

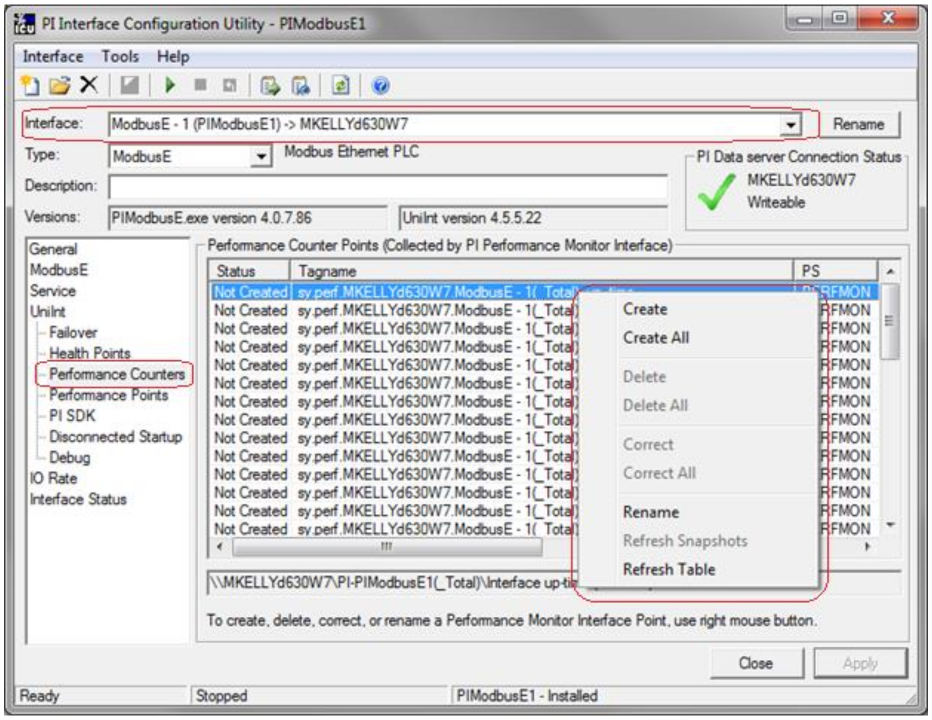
Status	Tagname	PS	Sr

Service PIPerfMon_2.0.2.91 is not installed on this machine.

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

Ready | Stopped | PIModbusE1 - Installed

After installing the PI 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:


PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 | UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Performance Counter Points (Collected by PI Performance Monitor Interface)

Status	Tagname	PS
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Total)	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON
Not Created	sy.perf.MKELLYd630W7.ModbusE - 1(_Tota	PERFMON

\\MKELLYd630W7\PI-PIModbusE1(_Total)\Interface up-ti

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

Ready | Stopped | PIModbusE1 - 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 PI Performance Monitor Interface – and not this Interface – is responsible for updating the values for the Performance Counters Points in the PI Data Archive. So, make sure that the PI 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 PI 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 points. (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.

“PI Status” (PI_Status)

The *.PI_Status* Performance Counters Point stores communication information about the interface and the connection to the PI Data Archive. If the interface is properly communicating with the PI Data Archive then the value of the counter is ‘0’. If the communication to the PI Data Archive goes down for any reason then the value of the counter will be ‘1’. Once the interface is properly communicating with the PI Data Archive 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** parameter and whose `Location1` attribute matches the **Interface 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 the PI. Data Archive 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 points 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 the PI Data Archive 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 point attributes is updated and the point is no longer a part of the interface configuration. For example, changing the point source, location 1, or Scan attribute can cause the point 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 min. 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 min. 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_60min* Performance Counters Point indicates the number of points that have not received a new value in the last 60 min. 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 min. 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 the PI Data Archive. This counter does not include the amount of time to send the data to the PI Data Archive. 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)

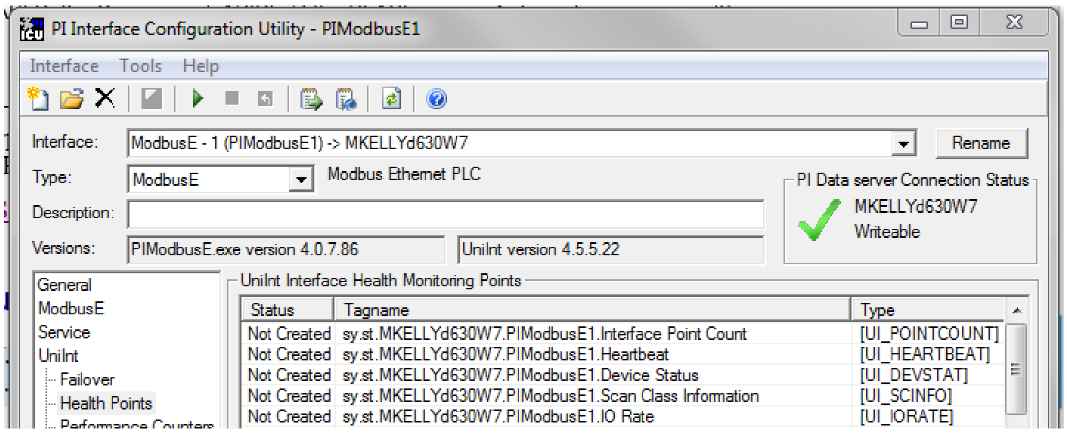
A *.scan_time* Performance Counter Point is available for each Scan Class of this Interface.

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 the PI Data Archive. 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 - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

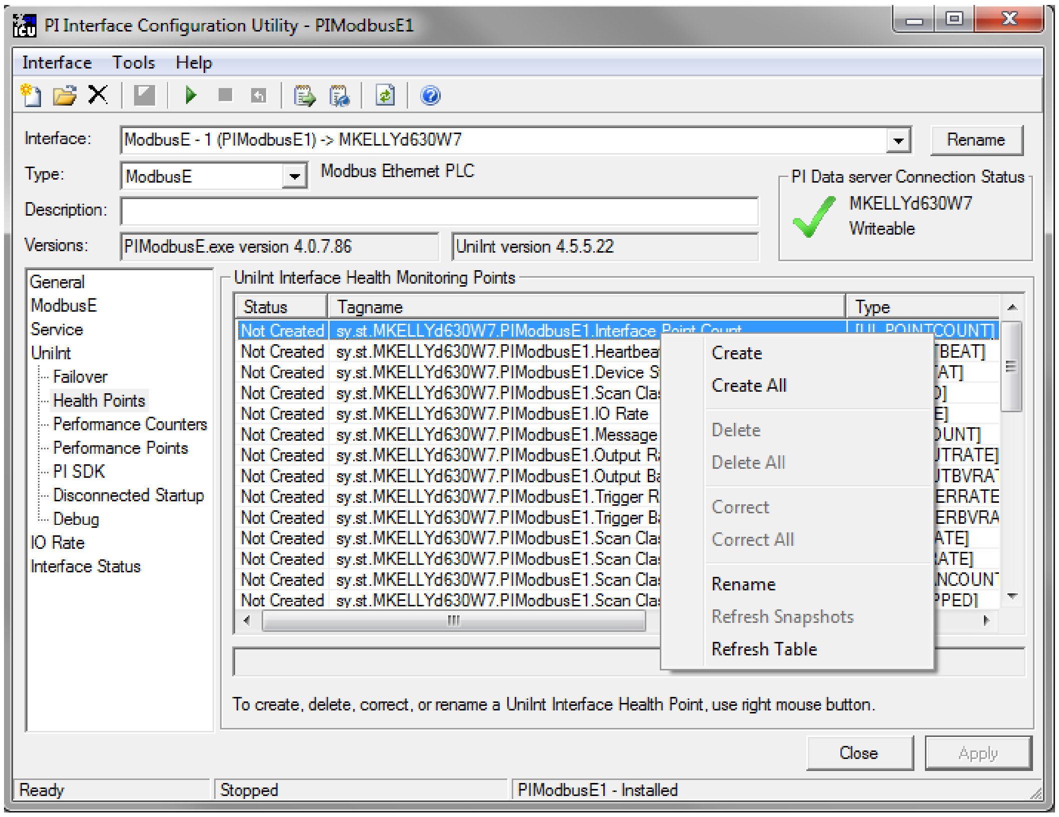
Versions: PIModbusE.exe version 4.0.7.86 — UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

UniInt Interface Health Monitoring Points

Status	Tagname	Type
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Interface Point Count	[UI_POINTCOUNT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Heartbeat	[UI_HEARTBEAT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Device Status	[UI_DEVSTAT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Class Information	[UI_SCINFO]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.IO Rate	[UI_IORATE]

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



PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE — Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 — UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

UniInt Interface Health Monitoring Points

Status	Tagname	Type
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Interface Point Count	[UI_POINTCOUNT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Heartbea	TBEAT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Device S	AT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Cla	)]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.IO Rate	E]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Message	OUNT]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Output R	JTRATE]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Output B	JTBVRA
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Trigger R	ERRATE
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Trigger B	ERBVRA
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Cla	ATE]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Cla	ATE]
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Cla	NCOUN
Not Created	sy.st.MKELLYd630W7.PIModbusE1.Scan Cla	PPED]

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

Ready | Stopped | PIModbusE1 - Installed

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]

A device status point is a type of interface Heath point. Specifically, it is a PI point that is updated by the interface to indicate the current interface working state. For example, if a device status point exists, the interface will send an update when it establishes or loses communication with Intellution. In this way, users can monitor the device status point to track the health of the interface without referring to log files.

A device status point must be a string point and the first characters in its ExDesc attribute must be [UI_DEVSTAT]. Refer to the *UniInt Interface User Manual* for more information on configuring interface Health points.

The following events can be written to the device status point:

- "1 | Starting" – UniInt writes this string to the Device Status point when the interface starts. The snapshot for the Device Status point will contain this value until either communication is established with Intellution on the local node or the interface shuts down.
- Digital state Good – the interface writes this event to the Device Status point when it establishes communication with Intellution on the local node.
- If the interface loses communication with the Intellution on the local node, the interface writes one of the following strings to the Device Status point:
    - "3 | 1 device(s) in error | Local Intellution stopped; interface shutting down."
    - "3 | 1 device(s) in error | Local Intellution stopped; interface will continue."
- If the interface is unable to collect alarm & event data it will write one of the following updates to the Device Status point;
    - "3 | 1 device(s) in error | Unable to collect alarm & event data."
    - "3 | 1 device(s) in error | Service library not loaded."

- "4 | Intf Shutdown" – UniInt writes this string to the Device Status point when the interface stops.

[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 points 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 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 log messages

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 PI points loaded by the interface. This count includes all input, output and triggered input points. This count does NOT include any interface health points 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 point if the point has a `SourceTag`. The [UI_OUTPUTRATE] Health Point tracks the number of these values. If there are no output points 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's. 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 points that have a `SourceTag`. If there are no output points 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's. 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 points. If there are no event-based input points 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's. 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 points. If there are no event-based input points 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's. 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 points in 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 points 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 points, and send the values to the PI Data Archive.

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 points.

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 PI Data Archive.

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 PI Data Archive, or elsewhere.

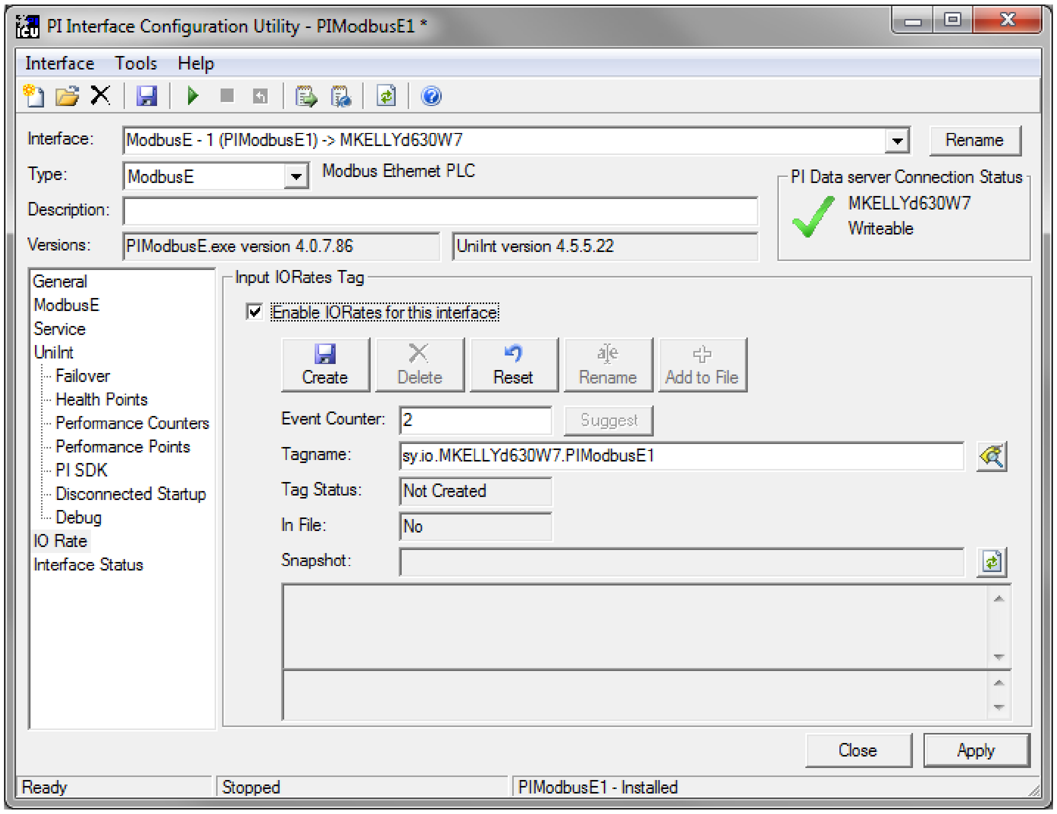
The Interface updates the value of this point at the completion of the associated scan.

I/O Rate Point

An I/O Rate point measures the rate at which the Interface writes data to its input points. 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 PI Data Archive.

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*.


PI Interface Configuration Utility - PIModbusE1 *

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE   Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86   UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Input IORates Tag

☑ Enable IORates for this interface

Event Counter: 2

Tagname: sy.io.MKELLYd630W7.PIModbusE1

Tag Status: Not Created

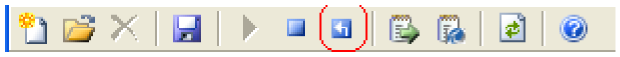
In File: No

Snapshot:

Ready | Stopped | PIModbusE1 - Installed

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:

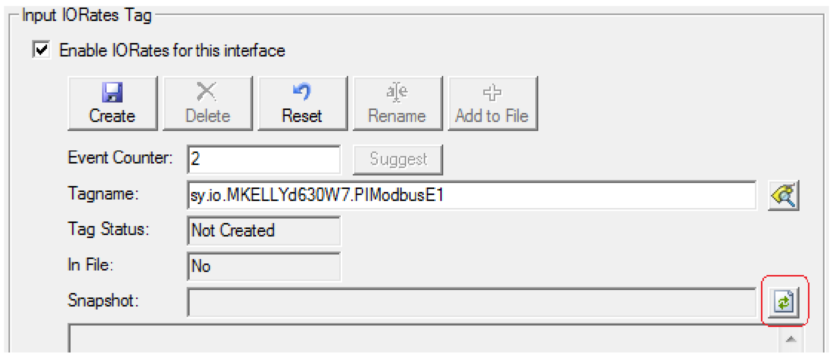


(The reason you need to restart the Interface is that the `PointSource` attribute of an I/O Rate point is `Lab`.)

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

```
PI-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:


Input IORates Tag

☑ Enable IORates for this interface

Event Counter: 2

Tagname: sy.io.MKELLYd630W7.PIModbusE1

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 point 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 point.

Tag Status

The *Tag Status* column indicates whether the I/O Rate point exists in the PI Data Archive. The possible states are:

- Created – This status indicates that the point exist in the PI Data Archive
- Not Created – This status indicates that the point does not yet exist in the PI Data Archive
- Deleted – This status indicates that the point has just been deleted
- Unknown – This status indicates that the PI ICU is not able to access the PI Data Archive

In File

The *In File* column indicates whether the I/O Rate tag in the *Tagname* and the number in the *Event Counter* box 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 point, if the I/O Rate point exists in the PI Data Archive. The *Snapshot* column is updated when the *IORates/Status Tags* tab is clicked, and when the Interface is first loaded.

Create/Save

Create the suggested I/O Rate point with the tag name indicated in the *Tagname* box. Or, save any changes for the tag name indicated in the *Tagname* box.

Delete

Delete the I/O Rate point listed in the *Tagname* box.

Rename

Change the tag name for the I/O Rate point listed in the *Tagname* box.

Add to File

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

Search

Search the PI Data Archive for a previously defined I/O Rate point.

Interface Status Point

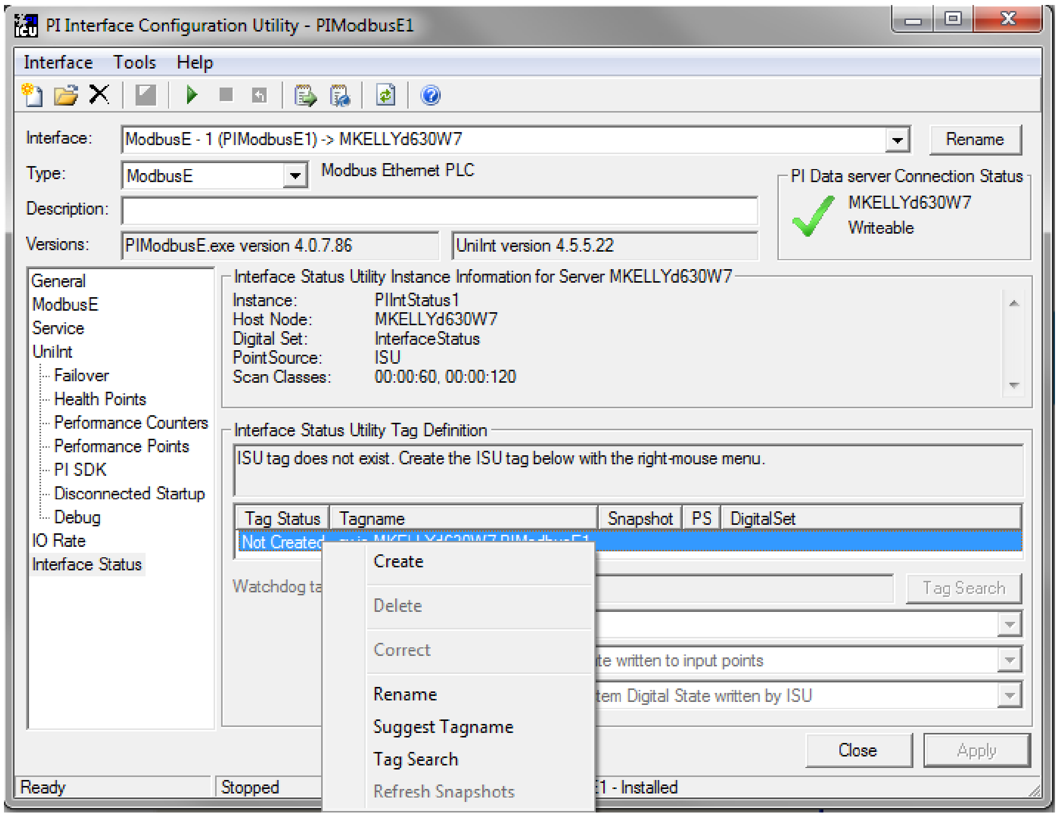
The PI Interface Status Utility (ISU) alerts you when an interface is not currently writing data to the PI Data Archive. This situation commonly occurs if

- the monitored interface is running on an interface node, but the interface node cannot communicate with the PI Data Archive; 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 Point. The watchdog point is a point whose value a monitored interface (such as this Interface) frequently updates. The watchdog point has its `excdev`, `excmin`, and `excmax` point attributes set to 0. So, a non-changing timestamp for the watchdog point indicates that the monitored interface is not writing data.

Please see the *Interface Status Interface* for complete information on using the ISU. PI Interface Status runs only on a PI Data Archive Node.

If you have used the ICU to configure the PI Interface Status Utility Interface on the PI Data Archive node, the ICU allows you to create the appropriate ISU point. Select this interface from the *Interface* list and click *Interface Status* in the parameter category pane. Right click on a point in the *Interface Status Utility Tag Definition* list to open the shortcut menu.


PI Interface Configuration Utility - PIModbusE1

Interface: ModbusE - 1 (PIModbusE1) -> MKELLYd630W7

Type: ModbusE Modbus Ethernet PLC

Description:

Versions: PIModbusE.exe version 4.0.7.86 UniInt version 4.5.5.22

PI Data server Connection Status: MKELLYd630W7 Writeable

Interface Status Utility Instance Information for Server MKELLYd630W7

Instance:	PIIntStatus1
Host Node:	MKELLYd630W7
Digital Set:	InterfaceStatus
PointSource:	ISU
Scan Classes:	00:00:60, 00:00:120

Interface Status Utility Tag Definition

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

Tag Status	Tagname	Snapshot	PS	DigitalSet
Not Created				

Ready | Stopped | 1 - Installed

Click *Create* to create the ISU point.

Use the *Tag Search* button to select a watchdog point. (Recall that the watchdog point 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 point. 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 point is `Incorrect`, right click to enable the context menu and select *Correct*.

Note: The PI Interface Status Utility – and not this Interface – is responsible for updating the ISU point. So, make sure that the PI 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.

For more information about logs for interfaces running on Windows, see *UniInt Interface Message Logging for UniInt 4.5.0.x and later Interfaces* or knowledge base article 401 on the OSIsoft technical support web site.

Messages are written to the log file at the following times.

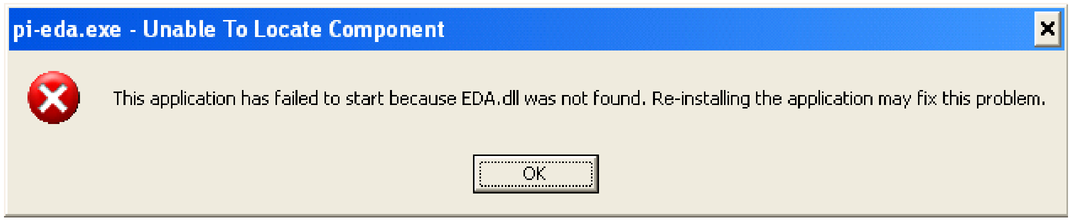
- When the Interface starts many informational messages are written to the log. These include the version of the interface, the version of UniInt, the command-line parameters used, and the number of points.
- As the Interface loads points, messages are sent to the log if there are any problems with the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

Messages

Interface-specific Troubleshooting

If the interface is behaving in an unexpected manner, check the log file. Even when the interface runs in interactive mode, not all error messages are written to the screen.

Interface Startup and Point-loading Errors


pi-eda.exe - Unable To Locate Component

This application has failed to start because EDA.dll was not found. Re-installing the application may fix this problem.

Check that the Windows Environment Variables (Control Panel -> System) contain the path to the `eda.dll` and `fixtools.dll` (assuming Intellution software is installed on the system).

Message	`EDA Failed to add tag [NODE, TAG, FIELD] to the group. NDK:Network Command Table (NCT) full.`
Meaning	Interface failed to initialize because the Intellution program `TCPTask.exe` has hung or is not running. Verify TCPTask is part of the startup list for the Intellution software. Restart the Intellution software and interface.

Message	`EDA Failed to add tag, [tagname], to the group. Location2 out of range.`
Meaning	Location2 defines whether the point is an input (0) or output (1). Verify the PI point definition has a Location2 value of either 0 or 1.

Message	`Complete NODE:TAG:FIELD information unavailable for PI tag: [tagname]`
Meaning	Verify that the complete Node, Tag, Field (NTF) definition has been defined. The point definition uses the InstrumentTag. If the NTF definition requires more than 32 characters, use the Exdesc point attribute to define the Node and Field parameters.

Message	`EDA Failed to add tag [NODE,TAG,FIELD] to the group. [tagname]`
Meaning	When debug is enabled for point checking, the interface attempts to verify the point with Intellution during startup. If this fails, it prints this message to the log file. Check the point configuration for this point, in particular the NTF definition. Launch the Intellution Database Builder program and verify you can view current values for this point.

Data Collection Errors

The following list of error codes describe the common return values the interface receives from Intellution when a point update request fails. They can be grouped into two general categories: network errors and non-network errors.

Non-network Errors

When the interface receives a non-network error from Intellution in response to a data request, it writes “Bad Input”, prints the error to the log file, and continues scanning for data. The point does not get dropped from the scan list (the interface will continue to try and read data for the point) but the error message will not be repeated in the log file. The message is only printed the first time the read fails.

Message	`Read failed. Error 1209 returned calling eda_get_float(); [tagname]`
Meaning	This error gets returned to the interface from Intellution on an update request and translates to “Illegal block field”. Verify the NTF definition (InstrumentTag) for the PI point configuration.

Message	`Read failed. Error 1212 returned calling eda_get_float(); [tagname]`
Meaning	This error gets returned to the interface from Intellution on an update request and translates to “Field’s value not known”. Verify that the Intellution software is currently scanning data for that point. Run Intellution Database Builder program and check that it is on scan and you can view a current value.

Message	*`Read failed. Error 1750 returned calling eda_get_float(); [tagname]`*
Meaning	This error gets returned to the interface from Intellution on an update request and translates to “Tag name is not defined”. Run Intellution Database Builder program and verify that the point exists on the defined node. Verify the NTF definition (InstrumentTag) for the PI point configuration.

Message	`Read failed. Eda_get_ascii returned empty string; [tagname]`
Meaning	When the interface gets a blank (null) value for a string or digital point, it writes ‘No Data’ to the PI point and logs this message to the log file.

Network Errors

When the interface receives a network error from Intellution, it writes “IO Timeout”, stops scanning for updates and goes into a wait loop while trying to re-establish a connection to Intellution.

Message	`Read failed. Error 1914 returned calling eda_get_float(); [tagname] [Node,Tag,Field]`
Meaning	This error gets returned to the interface from Intellution on an update request and translates to “Connection NOT established with node”. Verify the local Intellution software is running. If the point data is coming from a remote Intellution View/SCADA node check the network connection.

System Errors and PI Errors

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

Error Descriptions

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

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

UniInt Failover Specific Error Messages

Informational

Message	`16-May-06 10:38:00`
	`PI-EDA 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 PI Data Archive 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`
	`PI-EDA 1> UniInt failover: Interface in the "Primary" state and actively sending data to PI. Backup interface not available.`

Meaning	While in this state, the interface is in its primary role and sends data to the PI Data Archive 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`
	`PI-EDA 1> UniInt failover: Interface in the "Primary" state and actively sending data to PI. Backup interface available.`

Meaning	While in this state, the interface sends data to the PI Data Archive 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`
	`PI-EDA 1> One of the required Failover Synchronization points was not loaded.`
	` Error = 0: The Active ID synchronization point was not loaded.`
	`The input PI tag was not loaded`

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

Message	`16-May-06 17:38:06`
	`PI-EDA 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 PI tag was not loaded`

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

Message	`17-May-06 09:06:03`
	`PI-EDA > 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`
	`PI-EDA 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 1)

Message	`17-May-06 09:06:03`
	`PI-EDA 1> UniInt failover: Interface in an "Error" state. Could not read failover control points.`

Cause	The failover control points on the data source are returning a value to the interface that is in error. This error can be caused by creating a non-initialized control point on the data source." This message will only be received if the interface is configured to be synchronized through the data source (Phase 1).
Resolution	Check validity of the value of the control points on the data source.

Message	`16-May-06 17:29:06`
	`PI-EDA 1> Loading Failover Synchronization tag failed`
	`Error Number = 0: Description = [FailOver] or HeartBeat:n] was found in the exdesc for Tag Active_IN but the tag was not loaded by the interface.`
	`Failover will not be initialized unless another Active ID tag is successfully loaded by the interface.`

Cause	The Active ID or Heartbeat point is not configured properly. This message will only be received if the interface is configured to be synchronized through the data source (Phase 1).
Resolution	Check validity of point attributes. For example, make sure Location1 attribute is valid for the interface. All failover points must have the same PointSource and Location1 attributes. Modify point attributes as necessary and restart the interface.

Message	`17-May-06 09:05:39`
	`PI-EDA 1> Error reading Active ID point from Data source`
	`Active_IN (Point 29600) status = -255`

Cause	The Active ID point value on the data source produced an error when read by the interface. The value read from the data source must be valid. Upon receiving this error, the interface will enter the "Backup in Error state."
Resolution	Check validity of the value of the Active ID point on the data source. For example, use the Database Builder application to view and edit the value for the Active ID point.

Message	`17-May-06 09:06:03`
	`PI-EDA 1> Error reading the value for the other copy's Heartbeat point from Data source`
	`HB2_IN (Point 29604) status = -255`

Cause	The Heartbeat point value on the data source produced an error when read by the interface. The value read from the data source must be valid. Upon receiving this error, the interface will enter the "Backup in Error state."
Resolution	Check validity of the value of the Heartbeat point on the data source. For example, use the Database Builder application to view and edit the value for the Heartbeat point.

Errors (Phase 2)

Unable to open synchronization file

Message	`27-Jun-08 17:27:17`
	`PI 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.`
	`Initializing 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.

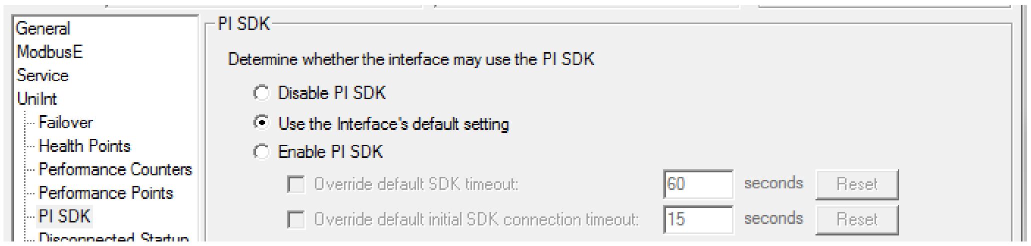
Error Opening Synchronization File

Message	`Sun Jun 29 17:18:51 2008`
	`PI 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 PI 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 PI Data Archive is not interrupted. If communication to the PI Data Archive 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 PI Data Archive version is earlier than 3.4.370.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**	**PI Data Archive earlier than 3.4.370.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. FIXtoPI Configuration Transfer Utility

Overview

A utility is provided to transfer configuration information contained in the Intellution database to points in the PI Data Archive. This utility must be considered as an aid rather than a total solution for configuring the PI Data Archive to work with the Intellution database.

The utility is a command line program called `FIXToPI.exe`.

The utility transfers the configuration information of the active raw data points in the Intellution database and formats them in a text file of appropriate commands for entry into the piconfig program.

The text file is named `FIXToPI.scr`, and it may be used in either of two ways. The first method is to run the piconfig utility with input redirected from this file. The second method is to use the `@INPUT` command of the piconfig utility.

The configuration transfer utility is designed to transfer information contained in Analog Input, Analog Output, Analog Register, Digital Input, Digital Output, Digital Register, and Multiple Digital Input blocks. If you want to archive information contained in other than those blocks, this must be done manually. In addition, the "Register" type blocks are configured as PI input points and thus will be read by the interface instead of being able to write to the Registers. If the client wants to configure "Register" type blocks as PI output points, the point must be edited manually in piconfig.

The utility must be run on a FIX SCADA node, as it uses FIX functions that will not work on a simple View node.

The program is designed to be flexible, allowing the transfer of all the information contained for the above type blocks as a default, and allowing you to restrict that transfer in a manner of your choosing.

- You can choose to allow the program to transfer configuration information from the SCADA node that the utility is running on and all the attached SCADA nodes, or you can choose to restrict it to any subset of those nodes.
- You can choose to allow it to transfer all tags of the types described above, or you can restrict that to any subset of those types.
- You can choose to allow transfer of all tags on the specified nodes, or you can exclude certain tagnames based on a simple pattern-matching scheme.
- You can also choose to only include tagnames that match a particular pattern. The pattern-matching scheme is simple – it is the one used in MS-DOS to match filenames; the '?' character matches any character, the '*' character matches all characters from that point on, and any other character is an exact match. Note that the pattern matching is case sensitive, so "ONE" is not the same pattern as "one".

The utility creates a unique digital set for each unique digital set in FIX when building the file to create the PI points. The digital set names assigned to the digital sets all start with the prefix dmFIXds. The suffix XXXX is appended where XXXX is a value from 0000 to 9999. The first digital set will be named dmFIXds0000, the second digital set will be named dmFIXds0001, etc. You should edit the digital state set names in the file where appropriate.

All digital output points are assigned a source tag with the same name as the tag name. This should be edited and the appropriate source tag name used.

User Instructions

The format of the command line for using the utility is:

```
FIXToPI /p=<pointsource> [/n=<node> [/n=<node>…]] [/t=<type> [/t=<type> …]] ^
      [/I=<include pattern> [/I=<include pattern> …]] ^
      [/e=<exclude pattern> [/e=<exclude pattern> …]]
```

Parameters

Parameter	Description
`/p=x` Required	The PI point source that you would like these points to have. This is a required parameter, and if not included, the program will exit with nothing done.
`/n=name` Optional	Name of a node. This parameter may be repeated for each node that the user wishes to include in the list of nodes. If no parameter of this type is specified, the program defaults to all nodes accessible by the machine on which the program is running.
`/t=type` Optional	Name of a block type. This parameter may be repeated for each block type that the user wishes to include in the list of block types. These can be any of "AI","AO","AR","DI","DO","DR","MDI, "AA", "DA". If no parameter of this type is specified, the default is to include all the above in the list of types.
`/e=exclusion_pattern` Optional	Pattern to match to the FIX block name to exclude from configuration transfer. The parameter may be repeated for each pattern the user wishes to exclude. If any of these types of parameters appears, the utility attempts to match each block name as encountered, and if the pattern match succeeds, the configuration information is NOT transferred. If multiple patterns are included, if the block name matches ANY of the patterns, the configuration information is NOT transferred.
`/I=inclusion_pattern` Optional	Pattern to match to the FIX block name to transfer configuration information. This parameter may be repeated for each pattern that the user wishes to include. If no parameter of this type is specified, the default is to include all the tags with the exception of the above exclude list. If one or more of these parameters are included, the configuration information is transferred for any block whose name matches any of the patterns in the list.

Note: Exclude processing is done before include processing, and therefore, if a block name matches the pattern of something on the exclude list, it will not be subjected to include list processing.

Sample Command Lines

To transfer all tags on node SCADA1:

```
FIXToPI /p=E /n=SCADA1
```

To transfer all the analog points for all connected SCADA nodes:

```
FIXToPI /p=G /t=AO /t=AI /t=AR
```

To transfer the Digital Input block information on node "LOCAL" with names beginning with the letters 'I' or 'J'

```
FIXToPI /p=x /I=I* /I=J* /n=LOCAL /t=DI
```

To transfer all configuration information of all blocks on all connected nodes except for the blocks with names containing a '1' as the first character, anything in the next two characters, "CHK" as the next three characters and anything after that.

```
FIXToPI /p=k /e=1??CHK*
```

Sample FixToPI.scr File

After the Utility has been run, the user should first edit the file FIXToPI.scr prior to creating the PI points and digital sets. The following example output shows the file that will be created in order to create a PI point for each FIX point type.

FIX Tag Name	FIX Point Type
AI1	AI
AO1	AO
AR1	AR
DI1	DI
DO1	DO
DR1	DR
MDI1	MD
AA1	AA (supported but not shown)
DA1	DA (supported but not shown)

Sample Output

Sample output from the utility is:

```
@table pids
@mode create, t
@istructure set,state,...
dmFIXds0000,OPEN,CLOSE
dmFIXds0001,OPENUP,CLOSEUP
dmFIXds0002,state0,state1,state2,state3,state4,state5,state6,state7
@endsection
@table pipoint
@ptclass classic
@mode create, t
@istructure
tag,pointsource,descriptor,pointtype,digitalset,ptaccess,dataaccess,
archiving,scan,instrumenttag,location1,location2,location4
DAVID:DI1,E,Digital Input 1,Digital,dmFIXds0000,o:rw g:rw w:rw,o:rw g:rw
w:rw,1,1,"DAVID,DI1,D_CV",1,0,1
DAVID:DO1,E,Digital Output 1,Digital,dmFIXds0001,o:rw g:rw w:rw,o:rw g:rw
w:rw,1,1,"DAVID,DO1,D_CV",1,1,1
```

```
DAVID:DR1,E,Digital Register 1,Digital,dmFIXds0000,o:rw g:rw w:rw,o:rw g:rw
w:rw,1,1,"DAVID,DR1,D_CV",1,0,1
DAVID:MDI1,E,,Digital,dmFIXds0002,o:rw g:rw w:rw,o:rw g:rw
w:rw,1,1,"DAVID,MDI1,M_CV",1,0,1
@endsection
@istructure tag,sourcetag
DAVID:DO1,DAVID:DO1
@endsection
@table pipoint
@ptclass classic
@mode create, t
@istructure
tag,pointsource,descriptor,pointtype,zero,span,typicalvalue,engunits,
excdev,excmin,excmax,compdev,compmin,compmax,ptaccess,dataaccess,archiving,
compressing,scan,instrumenttag,location1,location2,location4
DAVID:AI1,E,Analog Input 1,Float32,0.000000,100.000000,50.000000,,1.000000,
0, 600, 2.000000, 0, 28800,o:rw g:rw w:rw,o:rw g:rw
w:rw,1,1,1,"DAVID,AI1,F_CV",1,0,1
DAVID:AO1,E,Analog Output
1,Float32,0.000000,100.000000,50.000000,ao1,1.000000, 0, 600, 2.000000, 0,
28800,o:rw g:rw w:rw,o:rw g:rw w:rw,1,1,1,"DAVID,AO1,F_CV",1,1,1
DAVID:AR1,E,Analog Register
1,Float32,0.000000,100.000000,50.000000,,1.000000, 0, 600, 2.000000, 0,
28800,o:rw g:rw w:rw,o:rw g:rw w:rw,1,1,1,"DAVID,AR1,F_CV",1,0,1
@endsection
@istructure tag,sourcetag
DAVID:AO1,DAVID:AO1
@endsection
@bye
```

After the editing has been done, the last step is to use the text file generated by this utility to generate points in the PI Data Archive itself. There are two methods of doing this. The first involves standard input redirection, which means that you run the piconfig utility but, instead of accepting input from the keyboard, you redirect that input so that it comes from the file.

```
Piconfig < FIXToPI.scr
```

The second way of using this file is to use the `@INPUT` command of the piconfig command set. To do this, start the piconfig utility:

```
Piconfig
```

Then, at the command prompt, enter the command `@INPUT` followed by the file name:

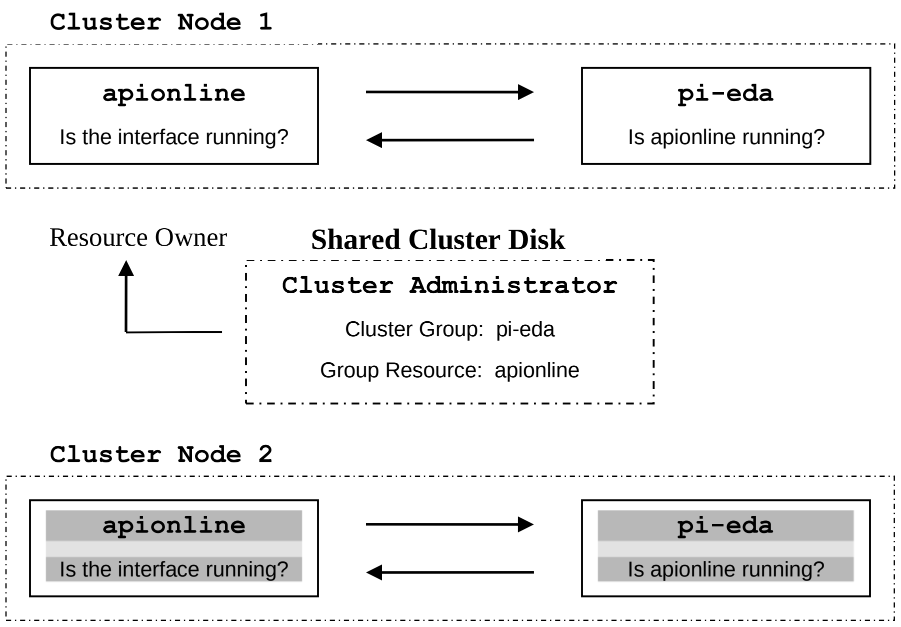
```
(Ls - ) Piconfig>@INPUT FIXToPI.scr
```

In both cases, ensure that you prepend the correct path information if this file is not in the current subdirectory.

Note: FixToPI utility is not a point auto-synchronization program. After it is run and changes are made later in FIX point database, it is your responsibility to check that the changes are still compatible with PI Data Arhive point attributes and, if necessary, the PI Data Archive point database is appropriately modified.

Appendix D. Cluster Failover

Principles of Operation



Cluster Failover Configuration Diagram

Interface-level failover is supported through Microsoft Cluster Services (MSCS). A cluster is composed of two or more nodes. Each member of the cluster has a copy of the interface installed and running, with only one node sending data to PI at any given time. Microsoft provides a Cluster Administrator program which is used for configuration and management of failover resources. A system failure (hardware or software) on the active cluster node will cause the Cluster Administrator to initiate a failover. On failover, ownership of a cluster resource is shifted from the node of failure to another available cluster node. In this way, it is ensured that only one cluster node owns the active interface at any given time.

Failover activity does not apply with respect to alarm/event message data collection. If enabled, alarm/event data will be collected on each interface node independent of cluster failover. However, it is strongly recommended that a separate copy of the interface be run specifically for collecting this type of data to maximize performance.

Setting up interface failover requires creating cluster groups and resources. These configurations are accomplished using the Cluster Administrator (see section Group and Resource Creation Using Cluster Administrator. The interface installation will distribute the program apionline into the install directory whose purpose is to run as a cluster group resource. On startup, the interface checks to see if the designated apionline cluster resource is running. If this is true, it tells the interface the local node owns the cluster group resource and is responsible for sending data to PI. Whichever cluster node owns the group resource is also the node where the active interface runs.

The apionline program serves two purposes: it indicates to the interface that it is currently active and it also prevents the Cluster Administrator from having an active node where the interface is not running.

The interface will query the Cluster Administrator to see if the apionline service is active. Since apionline is configured as a cluster group resource, it will only be active if the Cluster Administrator designates the local node as the group resource owner. In turn, when the apionline service is active, it checks to see that the interface service is running. If at any time the interface service terminates, apionline will shut itself down, thus initiating a failover. In this way, apionline prevents the Cluster Administrator from designating a node where the interface is not running to be owner of the cluster resource group.

The interface has the option of running in either warm or hot failover mode. Warm failover means an inactive interface will not request data updates from Intellution but otherwise functions normally (processing point edits, alarm/event data collection, etc.). Hot failover means an inactive interface will request data updates but does not send them to PI. The advantage of running in hot failover mode is you minimize the risk of missing data on failover. However, to minimize loading on inactive cluster nodes, we recommend running in warm failover mode.

The interface can be configured to operate with a preference for running on a particular cluster node. This is referred to as running with primary node bias. In this configuration the interface will attempt to run on the primary node whenever possible. This behavior may be preferred if one of the cluster nodes has proven to be more stable or otherwise performs better than the others.

The Intellution software must also be installed on each cluster node. Redundancy should be enabled on both nodes so they share the same point database. See Appendix E: FIX Redundancy and the PI IntFix Interface for detailed configuration.

Cluster Failover Configurations

Configuring APIOnline

The interface installation kit will distribute the apionline files (`apionline.bat` and `APIOnline.exe`) into the interface install directory. Configuring apionline is a three step process. The first step is to configure the `apionline.bat` file so it includes the name of the interface service used for failover. The second step is to install the apionline program to run as a service. The last step is to define apionline as a cluster group resource.

The name of the interface service is specified in the `apionline.bat` file. This file requires two parameter definitions. The first parameter is the name of the apionline executable file. The **`/proc`** parameter is used to define the interface service. For example, if the interface service is installed as PI-EDA and the apionline executable file is `APIOnline.exe`, the `apionline.bat` file would contain the following:

```
REM Sample apionline.bat
APIOnline.exe /proc=PI-EDA
```

Apionline uses the same parameters for each node it runs on. This means that you must have the same installation directory and executable file name on each cluster node. For example, if on one node the installation directory is:

```
c:\Program Files\pipc\interfaces\pi-eda\PI-EDA.exe
```

Then on the other cluster nodes, the installation directory, `pi-eda`, and interface name, `PI-EDA.exe`, must match. Here is an example of how this might look on another cluster node:

```
d:\pipc\interfaces\pi-eda\PI-EDA.exe
```

However, to keep things simple it is recommended that the same name and installation path be used across all systems.

The apionline application must also be installed as a service. Installing a program to run as a service is done from the command prompt at the path where the program resides. The following is an example of installing the apionline service:

```
d:\pipc\interfaces\pi-eda>apionline /install /depend tcpip
```

The `apionline.bat` and `APIOnline.exe` file should reside in the same directory. By default, these files are located in the interface install directory, however this is not required. After apionline has been installed as a service, the files should not be moved without first removing the service, then reinstalling the service after relocating the files. The following is an example of removing an installed apionline service:

```
d:\pipc\interfaces\pi-eda>apionline /remove
```

The final configuration step requires that a unique cluster group be created for each unique instance of apionline. Each group should have its own copy of apionline defined as a resource. Resources are moved between cluster nodes by group. See Group and Resource Creation Using Cluster Administrator for information on how to setup cluster group resources.

Running Multiple Instances of the Interface

Running multiple instances of the interface on each cluster node requires a unique instance of apionline for each instance of the interface. Each copy of apionline must also belong to a unique cluster group and be installed to run as a service. Running multiple instances of the interface is useful for tracking problems or for distributing interface loading.

To differentiate between copies of apionline, append an integer to the name. This integer gets passed to the corresponding interface through the **`/RN`** interface parameter. For example, to run two copies of the interface, two copies of apionline are needed on each cluster node. The following table displays a list of the files and configuration parameters required for each cluster node to run in this configuration:

Program Executable	Configuration File	Required Configuration Parameters *
apionline1.exe	apionline1.bat	apionline1.exe /proc=PI-EDA1
PI-EDA1.exe	PI-EDA1.bat	/FO /RN=1 /ID=1 *
apionline2.exe	apionline2.bat	apionline2.exe /proc=PI-EDA2
PI-EDA2.exe	PI-EDA2.bat	/FO /RN=2 /ID=2 *

This is not a complete listing of the necessary interface startup parameters to run the interface. Please see section Startup Command File for a complete listing and definition of the available parameters.

The final configuration step requires that a unique cluster group be created for each unique instance of apionline. Each group should have its own copy of apionline defined as a resource. MSCS moves resources between cluster nodes by group. See Group and Resource Creation Using Cluster Administrator for information on how to setup cluster group resources.

Buffering Data on Cluster Nodes

Buffering is fully supported on cluster nodes. In order to take advantage of buffering, `bufserv.exe` should be installed on all participating cluster nodes at the time of PI API installation. No special configurations are required to enable buffering on a cluster node. It should be noted that there is a risk of incurring a substantial amount of out-of-order data in the scenario where a failover occurs at a time when both interfaces are disconnected from the PI Data Archive (thus buffering data). Upon reconnection, each cluster node will send buffered data simultaneously, which will result in out-of-order data. This will cause the PI Data Archive to increase resource consumption, particularly the PI Archive Subsystem, as it attempts to process these out-of-order events. For a complete discussion about how to configure buffering, see section Buffering.

Group and Resource Creation Using Cluster Administrator

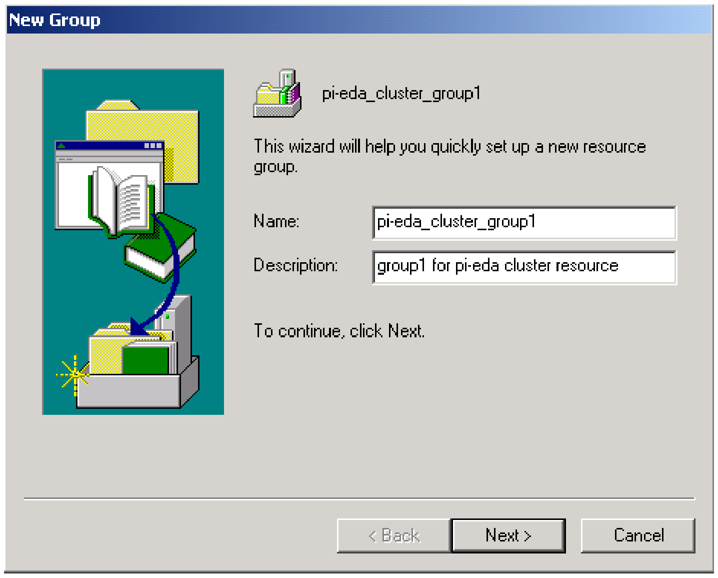
Before this step, make sure that MSCS is installed and configured. Test and verify that Clustering is functioning correctly prior to creating groups and resources for interface failover. Some steps for verifying correct cluster configuration are discussed at the end of this section. Apionline should also be installed and configured as described in section Configuring APIOnline.

Cluster Group Configuration

Note: Interfaces must not be run under the Local System account if you are using Cluster Failover. The service must be configured to run under an account that has administrator privileges.

Installation of Cluster Group

From the desktop, click **Start->Programs->Administrative Tools(Common)->Cluster Administrator**. Click **File->New->Group**. Enter the name of the group and description.



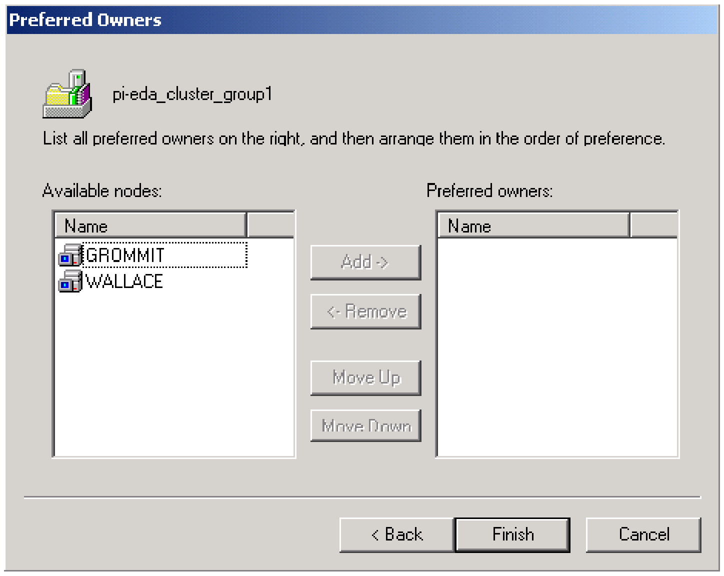
This wizard will help you quickly set up a new resource group.

Name: pi-eda_cluster_group1

Description: group1 for pi-eda cluster resource

To continue, click Next.

Click **Next**. Do not add any nodes to the **Preferred owners** box since owner preference is built into the interface through the cluster mode. Below, **Grommit** and **Wallace** are the cluster nodes.



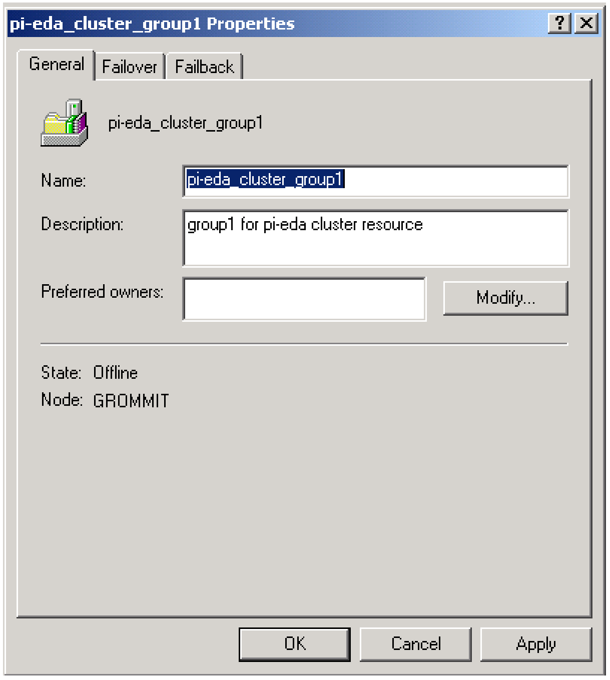
List all preferred owners on the right, and then arrange them in the order of preference.

Available nodes: Name
GROMMIT
WALLACE

Preferred owners: Name

Click **Finish**.

Right click the group you just created and select **Properties**. Fill out the name of the cluster and the description. Leave the **Preferred owners** box blank since these are the nodes on which you prefer the group to run. Preferred ownership is built into the interface through the cluster mode. Therefore you should not set this from the Cluster Administrator.


pi-eda_cluster_group1 Properties

pi-eda_cluster_group1

Name: pi-eda_cluster_group1

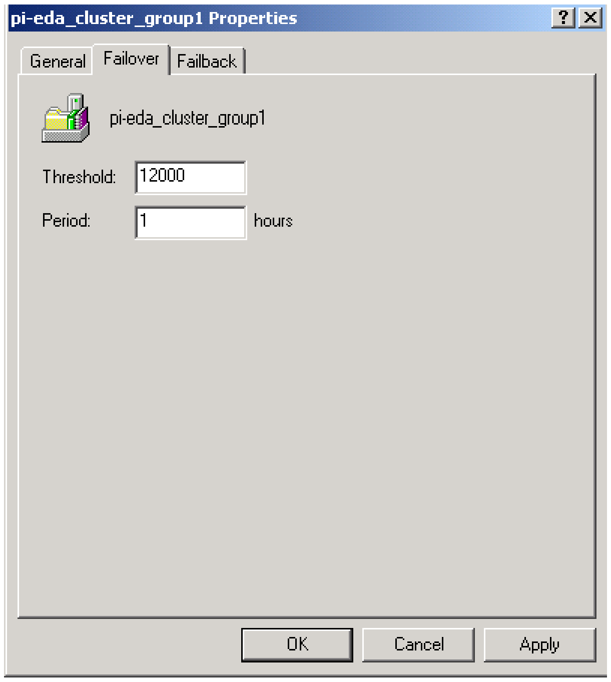
Description: group1 for pi-eda cluster resource

Preferred owners:

State: Offline

Node: GROMMIT

Set the Threshold and Period. Threshold is the maximum number of times you want to allow the group to fail over in the time specified by Period.

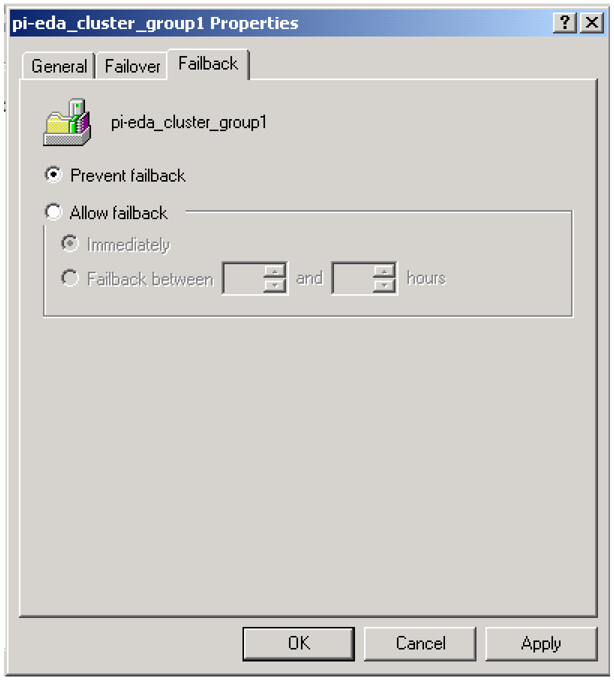

pi-eda_cluster_group1 Properties

pi-eda_cluster_group1

Threshold: 12000

Period: 1 hours

On the Failback tab, select **Prevent failback** because the failback mechanism is also built into the interface through cluster mode.


pi-eda_cluster_group1 Properties

pi-eda_cluster_group1

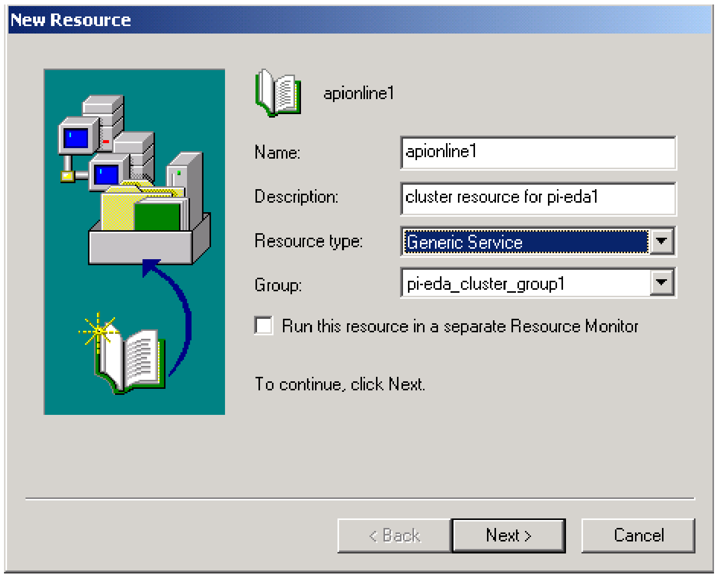
(● Prevent failback) ○ Allow failback

(● Immediately) ○ Failback between ___ and ___ hours

Click **Apply** and then **OK**.

Installation of the Resources

Right click the group in Cluster Administrator, select **New** and then **Resource**. Type the name of the resource and description. For **Resource type** select **Generic Service**.


New Resource

apionline1

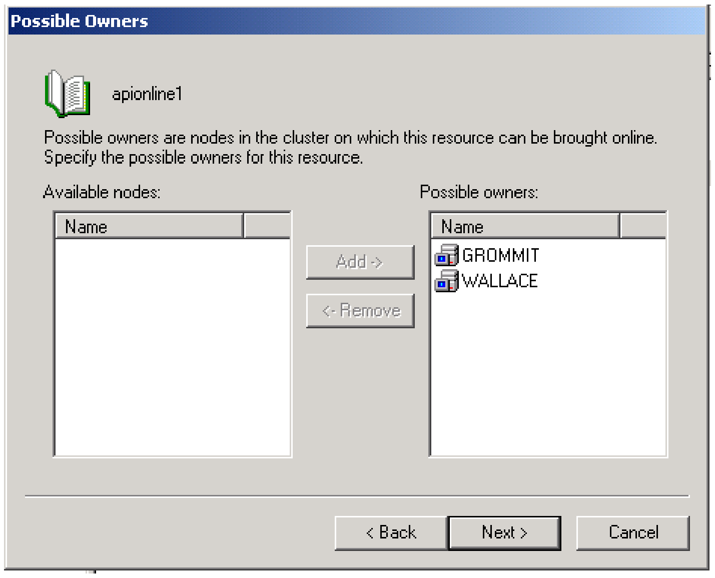
Name:	apionline1
Description:	cluster resource for pi-eda1
Resource type:	Generic Service
Group:	pi-eda_cluster_group1

☐ Run this resource in a separate Resource Monitor

To continue, click Next.

Running this resource in a separate Resource Monitor is not necessary unless this resource seems to be causing problems and you are trying to isolate the problem.

Click **Next** and verify that the cluster nodes are in the Possible owners list. These are the nodes on which the resource can run and, therefore, the nodes onto which the group can fail over.



apionline1

Possible owners are nodes in the cluster on which this resource can be brought online. Specify the possible owners for this resource.

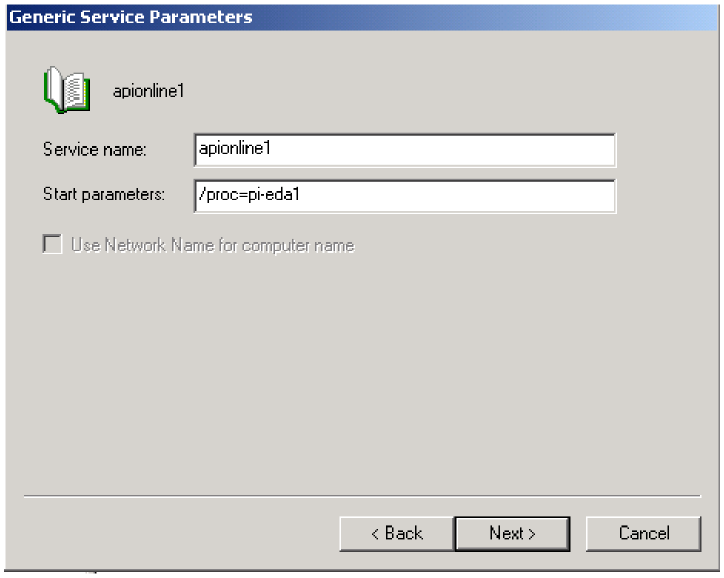
Available nodes:

Name

Possible owners:

Name
GROMMIT
WALLACE

Click **Next**, skip Dependencies, and continue with **Generic Service Parameters**.



apionline1

Service name: apionline1

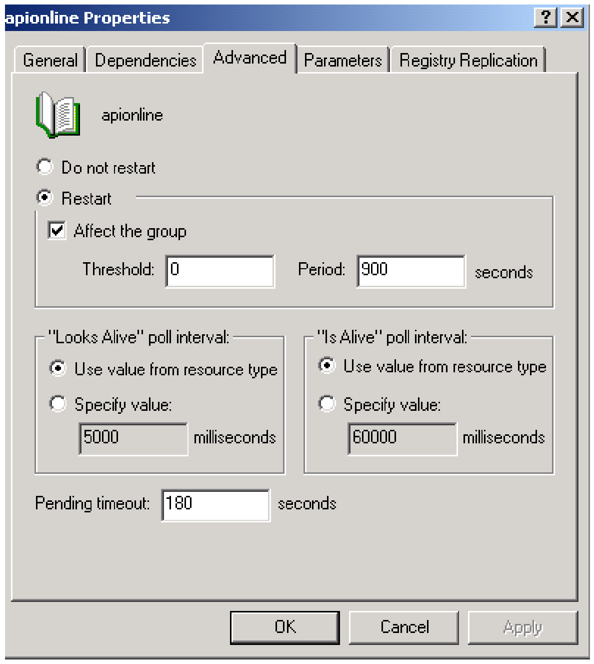
Start parameters: /proc=pi-eda1

☐ Use Network Name for computer name

The resource in the example above is called apionline1 and should have been installed as a service prior to cluster resource as described in the section Configuring APIOnline .

Click **Next** and skip Registry Replication. Click **Apply** and **OK**.

Right click the resource and select **Properties > Advanced** to set the entries as below. This indicates to MSCS to pass ownership of the resource to another cluster node before attempting to start it


apionline Properties

General | Dependencies | Advanced | Parameters | Registry Replication

apionline

Do not restart [✓ Restart]

[✓] Affect the group

Threshold: 0 Period: 900 seconds

"Looks Alive" poll interval: [✓ Use value from resource type] Specify value: 5000 milliseconds

"Is Alive" poll interval: [✓ Use value from resource type] Specify value: 60000 milliseconds

Pending timeout: 180 seconds

OK Cancel Apply

Click **Apply** and then **OK**.

Repeat the group and resource creation process for each instance of the interface on the node. Now you are ready to configure the interface.

Testing Cluster Configuration

The following configuration procedure can help identify any problems quickly. This is written for just one copy of the interface on each node. If configuring multiple copies, the first 5 steps are only needed for the first copy of the interface tested. When it says "matching" below, it means that `PI-EDA3.exe` looks for `apionline3.exe` and the apionline3 service and resource.

1. Configure the interface on each node with a dummy pointsource, one which is not currently used by any points, or with a **`PointSource`** and ID number that do not match the **`PointSource`** and **`Location1`** pair of any points. The goal is to bring up both interfaces with no points at all. Do not configure any failover-related parameters.
2. Start both interfaces and check the log to verify that both of them come up completely with no points. Any errors reported in the log must be corrected before continuing with the next step.
3. Use the Cluster Administrator to bring the matching cluster resource online by selecting the matching cluster group, then right-clicking on the resource and selecting **Bring Online**. Use the Task Manager to see that the matching apionline process is running on the node that Cluster Manager indicates owns the resource. For this configuration process, call that node OriginalOwner.

4. Still using Cluster Administrator, fail over the resource by selecting **Initiate Failure** in the right-click menu of the resource. You should see the resource state go to Failed and then Online Pending and then Online, with the other node now the owner. Depending on your system, you may not see the intermediate states, but you should see the resource end up Online with the other node as the owner. If not, you have a configuration problem and you must correct that before continuing the test.
5. Use Task Manager to verify that the matching apionline on the OriginalOwner node is no longer running and that the matching apionline service is now running on the other node (OriginalBackup node). If everything is good so far, move the resource to whichever node will be the primary node.
6. Now use Cluster Manager to take the resource Offline, then shut down both copies of the interface. Use the PI ICU to configure both interfaces for production. Do not forget to reset the **`PointSource`** and **`/ID`** to the correct values.
7. Bring up the interface on the node that does not currently own the group. The log should include:
   ```
   Cluster resource not online, state 4, waiting
   ```
8. Bring the resource online. The resource should failover to the node where the interface is running. After apionline is running on the same node as the interface, the log should include:
   ```
   Cluster Resource apionline1 on this node
   ```
   or possibly
   ```
   Resource now running on this node
   ```
9. Bring up the second interface. If the interface is configured with a cluster mode of primary node bias and the interface is currently running on the backup node, the resource will failover to the primary node. The log on the primary node will have one of the two messages listed in the last step.

Failover should now be configured correctly. Try failing the resource over a time or two, and shutting down one interface at a time to verify that the interfaces do what you expect.

Appendix E. FIX Redundancy and the PI IntFix Interface

Principles of Operation

Both FIX32 and iFIX support failover (starting from FIX32 version 6.15 and iFIX Dynamics version 2.0). The PI IntFix interface can take advantage of this functionality by running on a View node. A View node can look at a pair of SCADA nodes that have identical databases (and are connected to the same PLC) and obtain data from the currently active node. More information about Failover can be found in Intellution's documentation for FIX32 or iFIX. Although FIX allows a backup SCADA configuration that involves two SCADA servers and no View node, PI IntFix, as of version 2.4.0, does not support this configuration.

Note: iFIX does not synchronize the process databases on the SCADA servers. You must ensure that both databases are identical. It is also important that the failover-paired SCADA nodes' clocks are synchronized in order to ensure that the points get the same data regardless of which SCADA node the values are pulled from.

Note: FIX32 version 7.0 and iFIX 2.1 have been tested at OSIsoft for failover support and PI-EDA compatibility with FIX redundancy. The redundancy system tested consisted of pure FIX32 or pure iFIX combinations, i.e., two FIX32 SCADA nodes and one FIX32 View node, or two iFIX SCADA nodes and one iFIX View node. The average time the View node took to fail over from one SCADA node to the other was about 20-30 seconds. This is reflected in the data gap in the PI Archive.

This section describes the setup of the View node and the failover-pair SCADA nodes and PI point configurations so that PI IntFix can seamlessly collect data regardless of which SCADA node is active. Configurations are slightly different depending on whether the system is FIX32 or iFIX.

FIX32 Redundancy Setup

FIX32 View Node

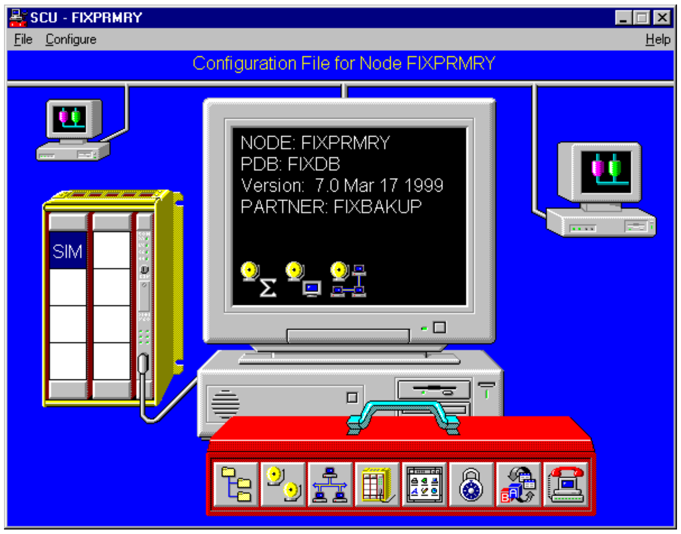
In the **Configure/Network** dialog box, enter remote node names with which the View node communicates.

Note: Only the primary node of the pair SCADA nodes needs to be entered here.

Click the **Configure** button.

Enter the backup node name for this remote SCADA node.

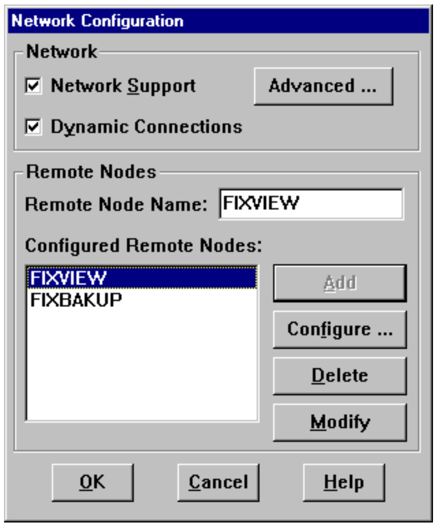
FIX32 Primary SCADA Node



NODE: FIXPRMRY
PDB: FIXDB
Version: 7.0 Mar 17 1999
PARTNER: FIXBAKUP

In the Configure/SCADA screen enter the database name. This database must reside both on the primary SCADA node and the backup SCADA node with the identical point definitions. Define Partner SCADA in Redundancy box.

Then in the **Configure/Network** dialog box,



Remote Node Name: FIXVIEW

Configured Remote Nodes:
FIXVIEW
FIXBAKUP

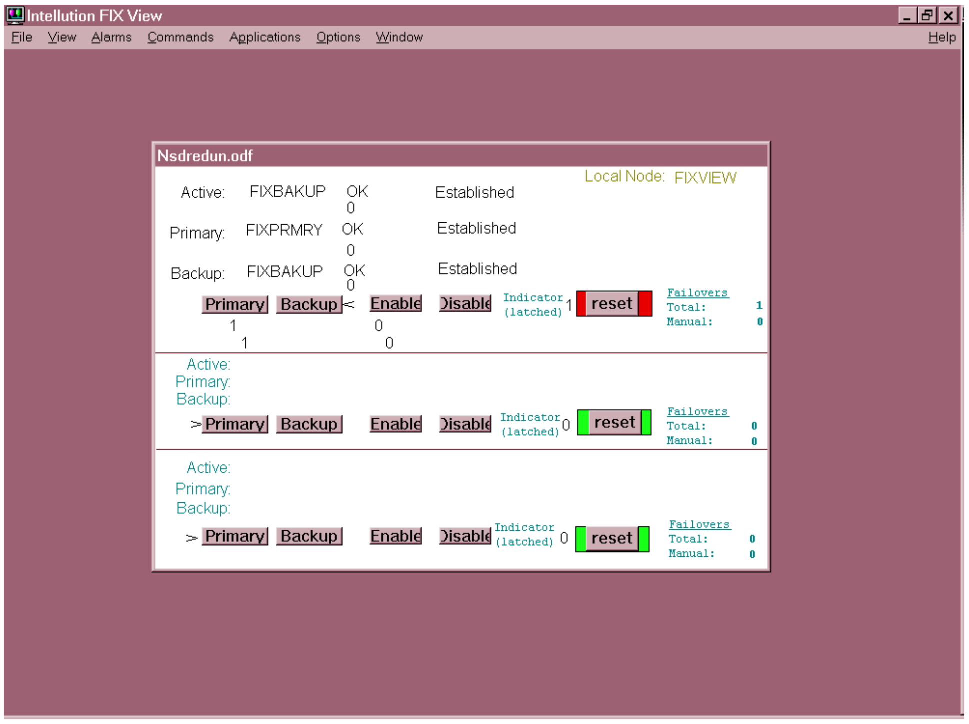
Enter the View node name and the SCADA partner node name in **Configured Remote Nodes** box. Then highlight the SCADA partner node, that is, this node's backup node, and click **Configure**. Enter its backup node's name in the redundancy box. Since this is the backup node's backup, it would be the primary node's name.

FIX32 Backup SCADA Node

Do the same thing as on the primary node, except that the local node name is the backup node name, Partner SCADA is the primary node, and the remote node's backup node is the backup node.

FIX32 View Node’s Network Status Display

In FIX View, when `nsdredun.odf` is opened the currently active SCADA node is displayed. This is the SCADA node from which the interface will be getting point values. For details about how to set this up, see the FIX32 documentation.



Local Node: FIXVIEW

Active: FIXBAKUP OK 0 Established

Primary: FIXPRMRY OK 0 Established

Backup: FIXBAKUP OK 0 Established

Indicator (latched) 1 — Failovers Total: 1 Manual: 0

Active:
Primary:
Backup:

Indicator (latched) 0 — Failovers Total: 0 Manual: 0

Active:
Primary:
Backup:

Indicator (latched) 0 — Failovers Total: 0 Manual: 0

FIX32 Node %windir%\system32\drivers\etc Host File

View node and both SCADA nodes must all have host files with the View node name, primary and backup SCADA node names, and IP addresses. For example,

```
xxx.xxx.xxx.1   FIXVIEW
xxx.xxx.xxx.2   FIXPRMRY
xxx.xxx.xxx.3   FIXBAKUP
```

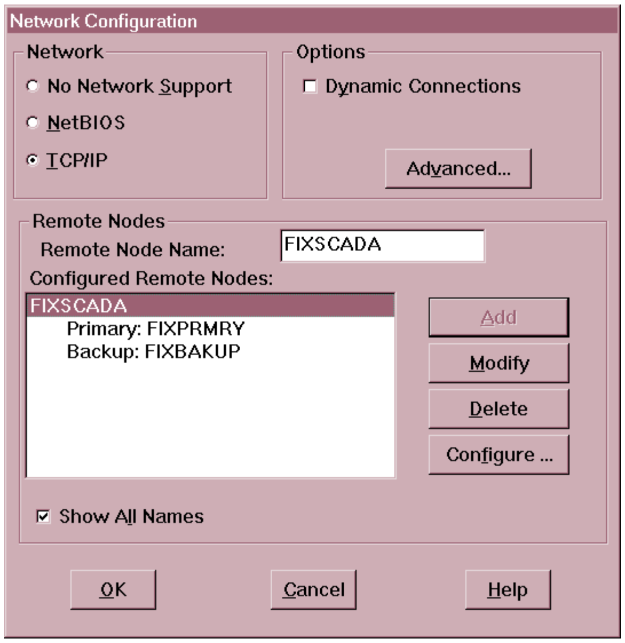
PI Point Configuration for FIX32 Tag

All configuration settings are the same as when no redundancy is required, except that the node name in the **`InstrumentTag`** attribute must be the primary node name.

iFIX Redundancy Setup

iFIX View Node

In System Configuration Utility (SCU) **Configure/Network** dialog box, enter the logical name that the View node is to communicate with in the **Remote Node Name** box and click **Add**. The logical node name appears in the **Configured Remote Nodes** list.


Network Configuration

Network: No Network Support / NetBIOS / [✓ TCP/IP]

Options: Dynamic Connections [ ]

Remote Nodes

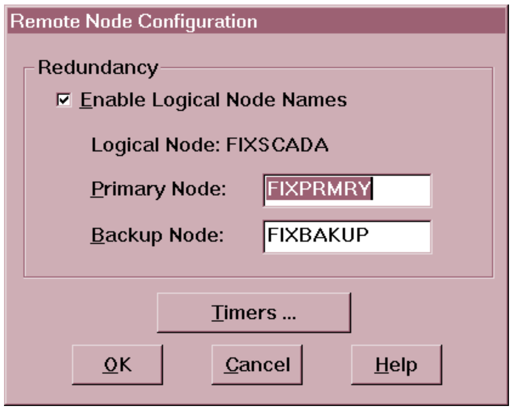
Remote Node Name: FIXSCADA

Configured Remote Nodes:

FIXSCADA
Primary: FIXPRMRY
Backup: FIXBAKUP

Show All Names [✓]

Click **Configure** and select **Enable Logical Names** check box. Enter the local node name of the primary node in the **Primary Node** box. Enter the local node name of the backup node in the **Backup Node** box.


Remote Node Configuration

Redundancy

Enable Logical Node Names [✓]

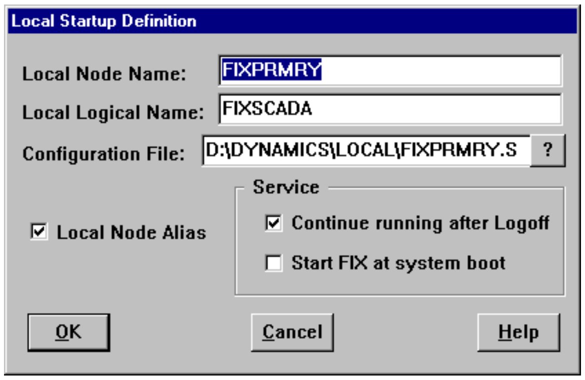
Logical Node: FIXSCADA

Primary Node: FIXPRMRY

Backup Node: FIXBAKUP

iFIX Primary SCADA Node

In the **SCU Configure/LocalStartup** dialog box, enter the logical name for the primary node and the backup SCADA pair.



Local Node Name: FIXPRMRY
Local Logical Name: FIXSCADA
Configuration File: D:\DYNAMICS\LOCAL\FIXPRMRY.S
☑ Local Node Alias
☑ Continue running after Logoff
☐ Start FIX at system boot

In the **SCU Configure/SCADA** screen enter the backup SCADA name.

Then access the **SCU Configure/Network** dialog box and enter the logical name for this SCADA and its backup SCADA node in the **Remote Node Name** box. Click **Add**.

Click **Configure** and select the **Enable Logical Names** check box. Enter the local node name of the primary node in the **Primary Node** box. Enter the local node name of the backup node in the **Backup Node** box.

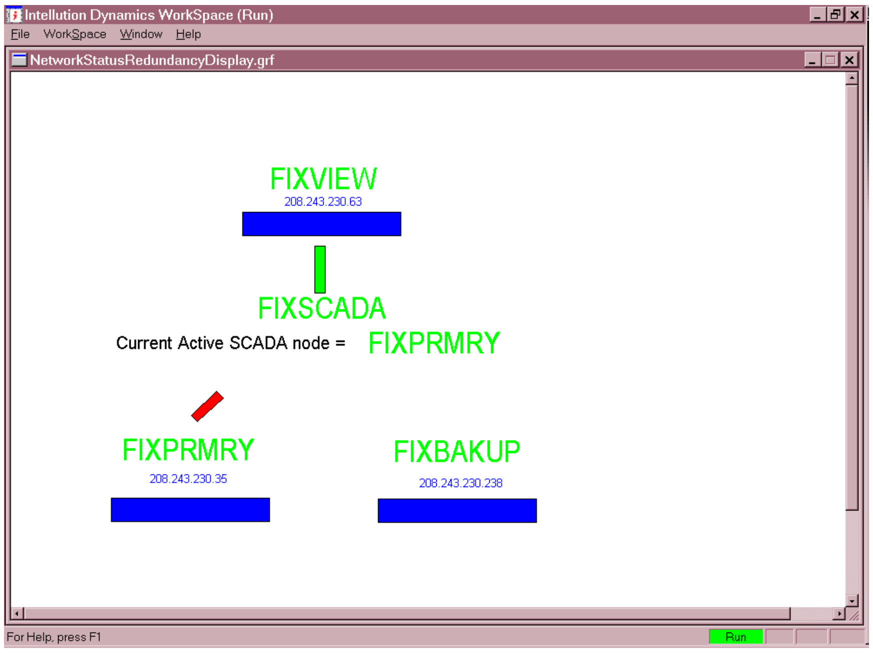
Return to the **SCU Configure/Network** dialog box and add the View node name in the **Remote Node Name** box.

iFIX Backup SCADA Node

Do the same as on the primary node, except that the local node name is the backup node name, and the Partner SCADA name in Configure/SCADA is the primary node name. The Configure/Network setting is identical to that of the primary SCADA node.

iFIX Network Status Redundancy Display

You can configure the `NetworkStatusRedundancyDisplay.grf` file to show which SCADA node is currently active. The interface gets its data from this active node.


Intellution Dynamics WorkSpace (Run)
File WorkSpace Window Help
NetworkStatusRedundancyDisplay.grf
FIXVIEW
208.243.230.63
FIXSCADA
Current Active SCADA node = FIXPRMRY
FIXPRMRY
208.243.230.35
FIXBAKUP
208.243.230.238
For Help, press F1
Run

iFIX Node %windir%\system32\drivers\etc Host File

View node and both SCADA nodes must all have host files with the View node name, primary and backup SCADA node names, and IP addresses. For example,

```
xxx.xxx.xxx.1   FIXVIEW
xxx.xxx.xxx.2   FIXPRMRY
xxx.xxx.xxx.3   FIXBAKUP
```

PI Point Configuration for iFIX Tag

All configuration settings are the same as when no redundancy is required, except that the node name in **`InstrumentTag`** attribute must be the logical SCADA node name for the failover, known as redundancy, SCADA pair.

Appendix F. OSI_iFIXmonitor Program

Introduction

Any program that uses the Intellution EDA library for iFIX, like this interface, can prevent iFIX itself from starting. This hazard is inherent in the implementation of the EDA library. After a program loads the EDA library and calls it, the EDA library acquires resources whose existence will prevent iFIX from starting if it is not already running. Once acquired, the resources held by the EDA library cannot be released programmatically and are only released when the program terminates. If iFIX stops while any programs that have called the EDA library are running, iFIX will refuse to restart until these EDA client programs terminate and consequently release the EDA library resources.

To avoid the situations that prevent iFIX from starting, an EDA client program must 1) wait until iFIX is known to be running before the EDA library is loaded or called, and 2) terminate if it detects that iFIX has shut down after the EDA library has been called.

The first requirement implies that an EDA client program must be able to determine whether iFIX is running without using the EDA library.

The second requirement is significant for an EDA client program that is a Windows service and needs to run continuously, like the PI IntFix interface. The second requirement implies that another program is needed to restart the service after it is obligated to terminate because iFIX has stopped.

The OSI_iFIXmonitor program, which is included in the interface installation kit, addresses these requirements for OSIsoft programs that use the EDA library: both the PI IntFix interface and PI AutoPointSync (PI APS) when any instances of the PI IntFix interface are registered for automatic point synchronization.

To operate correctly, OSI_iFIXmonitor must be configured as an iFIX task so that iFIX starts OSI_iFIXmonitor when iFIX itself starts. The section Configuring OSI_iFIXmonitor Program has instructions for configuring OSI_iFIXmonitor as an iFIX task.

OSI_iFIXmonitor registers with iFIX to receive notification before iFIX shuts down. When OSI_iFIXmonitor is notified that iFIX is about to stop, OSI_iFIXmonitor terminates. This is an oversimplification, as will be explained later in this appendix.

Because OSI_iFIXmonitor is started by iFIX and terminates before iFIX shuts down, a check for the existence of a running OSI_iFIXmonitor process can be used by other OSIsoft programs as an indication of whether iFIX is running. This method is independent of the Intellution EDA library.

As discussed earlier, any program that calls the EDA library must terminate when iFIX stops. In the case of programs that are Windows services and expected to run continuously, the services need to be restarted by some means. Since OSI_iFIXmonitor start up and shut down are coordinated with iFIX, OSI_iFIXmonitor is aware of events that affect these services. Therefore, several configurable options for controlling services have been built into OSI_iFIXmonitor. Specifically, OSI_iFIXmonitor can be configured to stop and optionally restart selected services when OSI_iFIXmonitor is notified that iFIX is about to stop.

OSI_iFIXmonitor Command-line Parameters

In its most basic mode, OSI_iFIXmonitor starts when iFIX starts and terminates when notified that iFIX is about to stop. The existence of a running OSI_iFIXmonitor process indicates that iFIX is running and, conversely, the absence of a running OSI_iFIXmonitor process indicates that iFIX is not running.

OSI_iFIXmonitor supports configurable options for controlling services that use the Intellution EDA library. The configuration of these options can be controlled by command-line parameters, loaded from the Windows registry, or both. OSI_iFIXmonitor merges the services configured in the registry with the services configured on the command line. If the same service is configured in both the registry and on the command line, the configuration options from the registry have precedence.

Since configuring an iFIX task requires manual interaction with the iFIX System Configuration Utility (SCU) and changes do not take effect until iFIX is stopped and restarted, OSI_iFIXmonitor is usually added to the iFIX task list once with no command-line parameters. OSI_iFIXmonitor loads configuration options from the registry, which can be changed at any time without requiring iFIX or OSI_iFIXmonitor to be stopped and restarted.

The PI ICU control for the IntFix interface and the PI APS configuration control for the Intfix_APS connector provide the means to configure OSI_iFIXmonitor registry settings for managing the respective services. PI ICU and the PI APS Configuration Utility are the recommended tools for configuring OSI_iFIXmonitor options.

In unusual situations, configuring OSI_iFIXmonitor by command-line parameters may be necessary. All command-line parameters are optional. The command-line parameters are listed in the table below.

All command-line parameters are case insensitive.

The leading / in most parameters is the Windows convention for switches. For consistency with UniInt, OSI_iFIXmonitor also recognizes leading – for switches.

OSI_iFIXmonitor processes options, left-to-right.

Parameter	Description
`servicename`	Any parameter that does not begin with a **/** or **–** is assumed to be the name of a service that OSI_iFIXmonitor manages. Multiple **`servicename`** parameters can be specified. All valid services are started when OSI_iFIXmonitor starts. Invalid service names will be rechecked for validity at shutdown. The delay before starting each service is controlled by the rightmost **`/delay`** parameter that precedes each **`servicename`** parameter. When notified that iFIX is stopping, the list of services from the registry is refreshed. All services in the list are revalidated. Finally, all valid services are stopped and

Parameter	Description
	optionally restarted. The action taken for each service during shutdown processing is controlled by the rightmost **`/stop`** or **`/restart`** parameter that precedes each **`servicename`** parameter.
`/stop`	When notified to stop, all services that follow this parameter (until superseded by a **`/restart`** parameter) are stopped and *not* restarted.
`/restart` **`Default`**	When notified to stop, all services that follow this parameter (until superseded by a **`/stop`** parameter) are stopped and then immediately restarted. This is the default handling for a service at shutdown.

`/delay=seconds`
`Default=0`
	When starting services during initial startup, delay the specified number of seconds before starting the services that follow this parameter.

`/reg[ister]`	All parameters are processed to determine the list of service names and the startup delay and shutdown handling applicable to each service. The configuration settings for the services are stored in the registry. No other processing is performed (no services are started or stopped).
`/unreg[ister]`	All parameters are processed to determine the list of service names. Configuration settings for these services are removed from the registry. No other processing is performed (no services are started or stopped).
`/monitor[debug]=list`	Enable logging of additional information from OSI_iFIXmonitor to the log file. The **`list`** for this parameter is a comma-separated list of the following type codes: Log all types. Equivalent to **`/monitor=3,4,5,6`**. Write log messages to stderr in addition to the log file. This code is only useful when OSI_iFIXmonitor is running in a command window. Log additional information while processing the command-line parameters. Log additional information about registry operations. Log additional information when starting services. Log additional information when stopping or restarting services.
`/client[debug]=list`	Enable logging of additional information from client programs (the PI IntFix interface or PI APS) to the log file. The **`list`** for this parameter is a comma-separated list of the following type codes: Log all types Equivalent to **`/client=2,3,4`**. Log additional information when dynamically loading or unloading the EDA library. Log additional information when searching for the OSI_iFIXmonitor process. Log additional information when checking the state of the OSI_iFIXmonitor process.

Appendix G. 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 PI Data Archives.

N-Way Buffering

If you have PI Data Archives that are part of a 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 PI Data Archives in a collective. (Bufserv also supports n-way buffering to multiple PI Data Archives in a collective; however, it does not guarantee identical archive records since point compression attributes could be different between PI Data Archives. With this in mind, OSIsoft recommends that you run PIBufss instead.)

ICU

ICU refers to the PI Interface Configuration Utility. The ICU is the primary application that you use to configure PI 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, OSIsoft discourages this approach. Instead, OSIsoft 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 PI 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
- PI Data Archive programs are not installed.

PI API

The PI API is a library of functions that allow applications to communicate and exchange data with the PI Data Archive. All PI interfaces use the PI API.

PI Data Archive Collective

A PI Data Archive Collective is two or more replicated PI Data Archives that collect data concurrently. Collectives are part of the High Availability environment. When the primary PI Data Archive in a collective becomes unavailable, a secondary collective member node seamlessly continues to collect and provide data access to your PI clients.

PI Data Archive Node

A PI Data Archive node is a computer on which PI Data Archive programs are installed. The PI Data Archive runs on the PI Data Archive node. In earlier documentation, PI Data Archive was referred to as the PI Server. (See PI Server Node.)

PIHOME

`PIHOME` refers to the directory that is the common location for PI 32-bit client applications.

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

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

PI 32-bit interfaces reside in a subdirectory of the `Interfaces` directory under `PIHOME`.

For example, files for the 32-bit Modbus Ethernet Interface are in

`[PIHOME]\PIPC\Interfaces\ModbusE`.

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

PIHOME64

`PIHOME64` is found only on a 64-bit operating system and refers to the directory that is the common location for PI 64-bit client applications.

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

PI 64-bit 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\PIPC\Interfaces\ModbusE`.

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

PI Message Log

The PI message log is the file to which OSIsoft interfaces based on UniInt 4.5.0.x and later write informational, debug and error messages. When a PI interface runs, it writes to the local PI message log. This message file can only be viewed using the PIGetMsg utility. See the Message Logs section for more information on how to access these messages.

PI SDK

The PI SDK is a library of functions that allow applications to communicate and exchange data with the PI Data Archive. Some PI interfaces, in addition to using the PI API, require the use of the PI SDK.

PI Server Node

In earlier documentation, the term “PI Server” was used as a nickname for the PI Data Archive and a PI Server node was a computer on which PI Data Archive programs were installed. While the PI Data Archive remains a core server of the PI Server product, the product name “PI Server” now refers to much more than the PI Data Archive. OSIsoft documentation, including this user manual, is changing to use “PI Server” in this broader sense and “PI Data Archive” to refer to the historian core. (See PI Data Archive Node.)

PI SMT

PI SMT refers to PI System Management Tools. PI SMT is the program that you use for configuring PI Data Archives. A single copy of PI SMT manages multiple PI Data Archives. PI SMT runs on either a PI Data Archive node or an interface node.

Pipc.log

The `pipc.log` file is the file to which OSIsoft interfaces based on UniInt versions earlier than 4.5.0.x write informational and error messages. When a PI 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 PI Data Archive. For a given timestamp, a PI point holds a single value.

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

Service

A 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 Point and Output Point)

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, PI System documentation uses the terms “tag” and “point” interchangeably.

Interfaces read values from a device and write these values to an input point. Interfaces use an output point to write a value to the device.

Appendix H. Technical Support and Resources

For technical assistance, contact OSIsoft Technical Support at +1 510-297-5828 or techsupport@osisoft.com. The OSIsoft Technical Support website offers additional contact options for customers outside of the United States.

When you contact OSIsoft Technical Support, be prepared to provide this information:

- Product name, version, and build numbers
- Computer platform (CPU type, operating system, and version number)
- Time that the difficulty started
- Log files at that time
- Details of any environment changes prior to the start of the issue
- Summary of the issue, including any relevant log files during the time the issue occurred

The OSIsoft Virtual Campus (vCampus) website has subscription-based resources to help you with the programming and integration of OSIsoft products.

Appendix I. Revision History

Date	Author	Comments
21-Oct-97	MH	First draft
22-Oct-97	MH	First version reviewed
11-Nov-97	DM	Data type revisions
12-Nov-97	DM	Local failure detection
23-Jan-98	DM	Added configuration transfer utilities
09-Apr-98	DM	Offloaded NTF components to ExDesc field in PI
15-Apr-98	JFZ	Re-added utility info from version 1.3 manual.
13-May-98	DM	Additional information on string tags
16-Jun-98	Holly	Fixed table of contents, page nums were all listed as 0
17-Jul-98	Kyong-Ri	Noted changes since version 1.4.
08-Aug-98	Kyong-Ri	Corrected descriptions in /L switch and logging tag sections. Deleted Logging Tag section. Modified /L description to reflect the change in code which causes the interface to abort instead of hanging (v1.8).
10-Sep-98	Kyong-Ri	Corrected the error in manuals up to version 1.8.2 regarding the delimiter after event=xxxx entry in the PI extended descriptor. Now both ‘,’ and ‘;’ are allowed to end NODE name and FIELD name. However, a comma must still be used to end event tag name.
03-Dec-98	Kyong-Ri	Added a more detailed message list under Trouble-shooting section. Added comments on added features (optional local server time switch).
29-Mar-99	Kyong-Ri	Added descriptions of enhancement features.
15-Apr-99	Kyong-Ri	Modified explanation for eda error 1212. Included debug symbol installation instructions. Added descriptions of more debug switches in command line.
03-Aug-99	Kyong-Ri	Added explanation for new data type support (FIX float to PI digital mapping) .
25-Jan-00	Kyong-Ri	Added FIX redundancy information.
03-Jul-00	Kyong-Ri	Corrected Location1 range from 1 to 99 to 0 to 98.
28-Jul-00	Kyong-Ri	Added more comments about user queue in /qn section
09-Aug-02	Pwilliams	Updated for alarm/event msg data collection. Changed manual format to meet new standard.
2-Oct-02	Chrys	Updated to skeleton 1.11

Date	Author	Comments
21-Apr-03	Pwilliams	Added Appendix C: Cluster Failover. Included failover parameters in startup file section.
11-Jun-03	Pwilliams	Updated version on title page to 2.1.3.0
13-Jun-03	Pwilliams	Updated for new ICU, incremented version to 2.2.0. Moved FIX redundancy section to Appendix D. Revised command line startup switches section.
18-Jun-03	Pwilliams	Revised Point Attribute, Principles of Operation & Supported Features sections. Revised Hardware Diagram. Removed Logging section from Appendix A (now supported through debug startup switch). Revised Error Messages in Appendix A.
19-Jun-03	Pwilliams	Added alarm/event message data and redundancy subsections to Principles of Operation. Added diagrams for redundancy architecture to Principles of Operation section.
16-Sep-03	Pwilliams	Updated version on title page – no other changes.
16-Mar-04	Pwilliams	Incremented version on title page. Updated troubleshooting section for cases resulting in ‘No Data’.
29-Jul-04	Pwilliams	Incremented version on title page to 2.2.1.1. Fixed typo in Appendix C for /FM switch.
19-Oct-04	Chrys	Version 2.2.0.0 – 2.2.1.1 Rev B: Removed triplicate descriptions of command-line parameters; fixed headers and footers; fixed section breaks; added platforms to intro table
27-Oct-04	Pwilliams	Version 2.2.0.0 – 2.3.0.1 Rev A: Incremented version on title page. Added supported output PI point types to InstrumentTag section table. Added discussion about sub-second timestamps below table of supported features.
2-Dec-04	MPKelly	Updated to latest manual skeleton.
22-Nov-05	Chrys	Version 2.2.0.0 – 2.3.0.1 Rev B: Changed name of interface from PI-EDA to PI IntFix; updated TOC.
19-Sep-06	Janelle	Version 2.2.0.0 – 2.3.0.1 Rev C: Updated Supported Features table to include APS connector; fixed headers and footers; updated How to Contact Us page.
17-Nov-06	Prowe	Version 2.3.2.45, Rev D; Updated manual to Skeleton v2.5.3, applied template and spell checked document.
5-Dec-06	Mkelly	Version 2.3.2.45, Rev E; Fixed headers and footers.
20-Feb-2007	Ldaley	Version 2.4.0.0, Rev A: Added new features for coordinating interface execution with Intellution.
27-Mar-2007	Pwilliams	Version 2.4.0.0, Rev B: Updated data redundancy, UniInt features, Location5 & Convers implementations.
18-Apr-2007	Mkelly	Version 2.4.0.0, Rev C: Updated Configuring the Interface using the PI ICU, added new screen shots and updated the TOC, fixed headers and footers.

Date	Author	Comments
18-Apr-2007	Pwilliams	Version 2.4.0.0, Rev D: Fixed labeling of screenshots, updated failover tables, updated Location5 usage.
22-May-2007	Janelle	Version 2.4.0.0, Rev E: updated hardware diagrams; update ICU screen shots for Cluster Failover
25-May-2007	Mkelly	Version 2.4.0.0, Rev F: Added /UHT_ID=# to the command-line parameter table.
18-Sep-2008	Pwilliams	Incremented version to 2.4.3.0.
09-Apr-2009	Pwilliams	Incremented version to 2.5.0.0. Updated configuration of string tag for all alarm and event data collection. Added description for interface specific behavior with UniInt Phase 2 failover. Migrated to interface skeleton 3.0.10. Updated ICU screen shots.
13-Apr-2009	Mkelly	Version 2.5.0.0 Revision A; Fixed headers and footer, screenshots, hyperlinks, support features table, and other formatting problems.
01-May-2009	Pwilliams	Incremented version to 2.5.4.0.
28-Feb-2011	Sbranscomb	Version 2.5.4.0 Revision A; Updated to skeleton version 3.0.31
10-Mar-2011	Pwilliams	Version 2.6.0.x, Updated supported platforms. Update alarm data collection section to mention support for out of order data.
19-Jul-2011	MKelly	Version 2.6.0.x – 2.6.1.x; Upped the version number for rebuild with new UniInt 4.5.2.0.
22-May-2012	DZhang	Added iFIX 5.5 to compatibility testing list
28-May-2013	DZhang	In Supported Features, updated 64-bit OS support from No to Yes. This has been true since version 2.6.1.26a.
23-Oct-2014	MKelly	Version 2.6.0.x – 2.6.1.x; Updated to skeleton 3.0.39, updated all ICU Control screenshots, fixed hyperlinks..