PI Interface for GSE D/3 DBA

Version 4.31.7.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

PI Interface for GSE D/3 DBA

Published: 09/2014

Table of Contents

Chapter 1. Introduction

The PI Interface for GSE D/3 DBA, also known as the PI GSETID3 Interface provides two-way communication with a NovaTech D/3 distributed control system. The interface program reads the PI point database to determine which points to read from and write to the NovaTech Configurator/Display Control Modules (CDCM/DCM).

The interface requires D/3 DBA software, versions 10.0, 10.1, 10.2, 11.2, 12.1, 12.2, 12.2-3, 14.0. 14.1, and 14.1-2 are supported.

Note: Please contact OSIsoft if a version of D/3 software other than one of those listed above is needed.

The interface can run on the various versions of Windows, but is limited to the versions on which the DBA D/3 software runs.

The PI System can run on the same computer as the D/3 software. The PI system can also run on another computer with the D/3 computer as a PI Interface node. In this way, it is possible to have several D/3 systems on different computers communicating with a PI System.

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: Throughout this manual there are references to where messages are written by the interface which is the PIPC.log. This interface has been built against a of UniInt version (4.5.0.59 and later) which now writes all its messages to the local PI Message log.

Please note that any place in this manual where it references PIPC.log should now refer to the local PI message log. Please see the document *UniInt Interface Message Logging.docx* 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 manual*
- *UniInt Interface User Manual*

Vendor

Consult the following NovaTech documentation for more information on D/3 software:

- *D/3 Database Access User's Guide (UG1.0007.V6.2C)*
- *D/3 PCM Database Fields Reference Manual, (RM1.0008.V6.2C)*

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	32-bit OS	No	No
	64-bit OS	No	No
Windows 2012	64-bit OS	No	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.

Please contact OSIsoft Technical Support for more information.

Supported Features

Feature	Support
Interface Part Number	PI-IN-GSE-D3DB-NTI
Auto Creates PI Points	No
Point Builder Utility	No
ICU Control	Yes
PI Point Types	Real / Digital / Integer / Float32 / Float16 / String
Sub-second Timestamps	No
Sub-second Scan Classes	No

Feature	Support
Automatically Incorporates PI Point Attribute Changes	Yes
Exception Reporting	Yes
Outputs from PI	Yes
Inputs to PI:	Scan-based
Supports Questionable Bit	No
Supports Multi-character PointSource	Yes
Maximum Point Count	Unlimited
* Uses PI SDK	No
PINet String Support	Yes
* Source of Timestamps	PI Data Archive
History Recovery	No
* UniInt-based * Disconnected Startup * SetDeviceStatus	Yes Yes Yes
* Failover	UniInt Failover (Phase 1, Phase 2: warm, cold)
* Vendor Software Required on Interface Node / PINet Node	Yes
Vendor Software Required on Foreign Device	No
Vendor Hardware Required	No
Additional PI Software Included with interface	No
Device Point Types	See Below
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.

Source of Timestamps

The source of timestamps is the PI Data Archive.

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

Disconnected Start-Up

The PI GSETID3 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 *UniInt Interface User Manual* for more details on UniInt Disconnect startup.

SetDeviceStatus

The NovaTech D/3 Interface is built with the UniInt version that supports device status tags. This interface supports health tags. The Health tag with the point attribute Exdesc = [UI_DEVSTAT] represents the status of the source device. The following events can be written into this tag:

- "1 | Starting" - the interface is starting.
- "Good" - the interface is properly communicating and reading data from the NovaTech D/3 DBA.
- The following event represents a failure to communicate with the NovaTech D/3 DBA (the system is not running):

  "3 | 1 device(s) in error | Interface is not Connected"
- "4 | Intf Shutdown" – the interface is stopped.

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

Failover

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

D/3 Database Access (DBA) software from NovaTech is required on the interface node. The interface requires D/3 DBA version 10.0, 10.1, 10.2, 11.2, 12.1, 12.2, 12.3, 14.0, 14.1 and 14.1-2.

Device Point Types

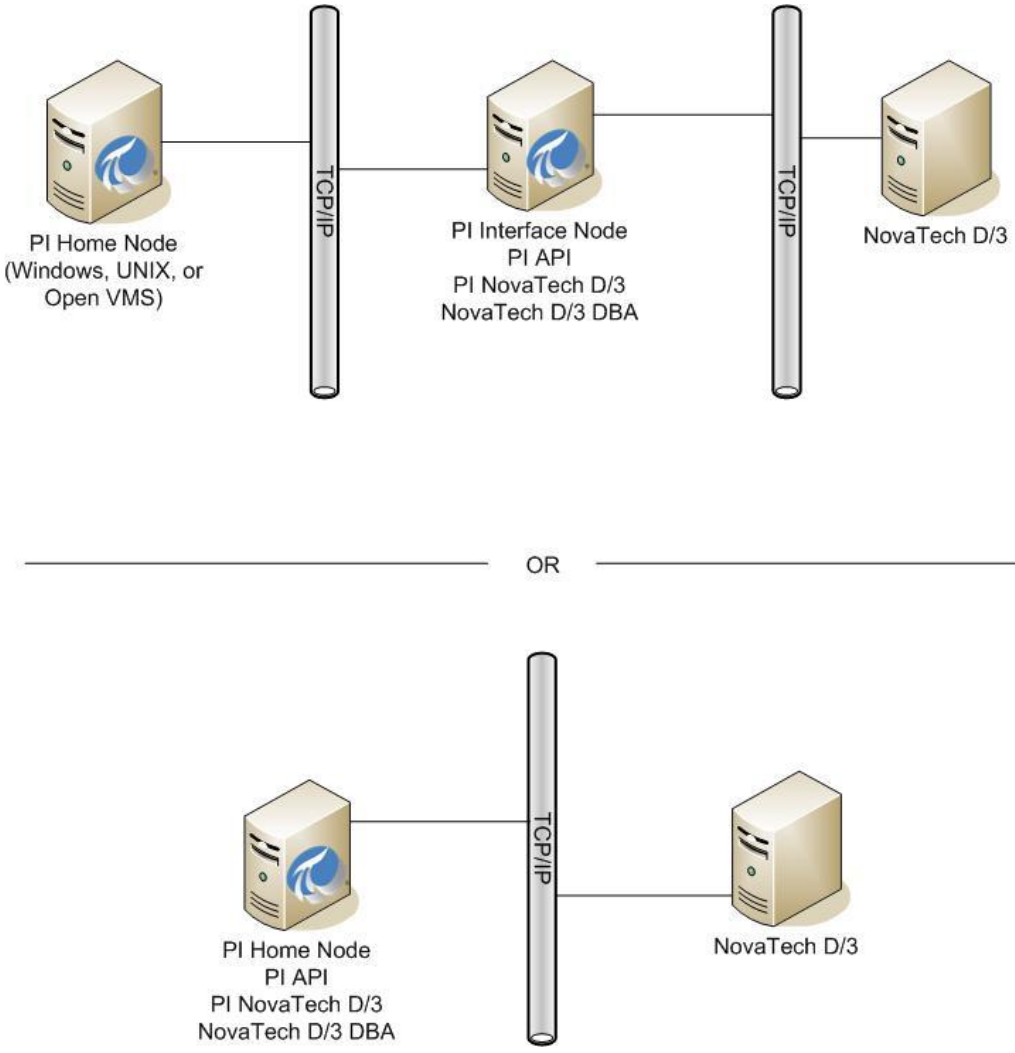
AI	CAL	CB	DED	DEV
AI_ALMCT	CAL_VAHI	CB_AFLG	DED_DIVL	DEV_ALMCT
AI_CRIT	CAL_VALO	CB_FLG	DED_DIVS	DEV_CMD
AI_CURAL	CAL_VAVL	CB_FLG1	DED_VEVL	DEV_DFLG
AI_DEAD	CAL_VAVS	CB_FLG2		DEV_FLG
AI_DEV	CAL_VBVL	CB_INVL		DEV_MODE
AI_DVLIM	CAL_VBVS	CB_INVS		DEV_STAT
AI_FLG	CAL_VCVL	CB_LOUT		DEV_TMP0
AI_HHLIM	CAL_VCVS	CB_OTHI		DEV_TMP1
AI_HLIM	CAL_VDVL	CB_OTLO		
AI_INVL	CAL_VDVS	CB_OTRT		
AI_INVS	CAL_VEVL	CB_OTVL		
AI_LLIM	CAL_VEVS	CB_OTVS		
AI_LLLIM	CAL_VFVL			
AI_MEAS	CAL_VFVS			
AI_RLIM				

DGR	DIN	DOT	FAN	INT
DGR_ALMCT	DIN_ALMCT	DOT_ALMCT	FAN_OVL1	INT_ETIM
DGR_FLG	DIN_FLG	DOT_FLG	FAN_OVS1	INT_K
DGR_VAL	DIN_VAL	DOT_VAL	FAN_LO1	INT_TOT
			FAN_OVL2	
			FAN_OVS2	
			FAN_LO2	

PRF	RMP	RB	SAV	SEL
PRF_FDSP	RMP_ICVL	RB_IBHI	SAV_I0HI	SEL_I0HI
PRF_FDVL	RMP_ICVS	RB_IBLO	SAV_I0LO	SEL_I0LO
PRF_FDVS		RB_IBVL	SAV_I0VL	SEL_I0VL
PRF_FDZE		RB_IBVS	SAV_I0VS	SEL_I0VS
PRF_GNVL		RB_INTRT	SAV_I1VL	SEL_I1VL
PRF_GNVS		RB_OBHI	SAV_I1VS	SEL_I1VS
PRF_GPAC		RB_OBLO	SAV_I2VL	SEL_I2VL
PRF_INISP		RB_OBVL	SAV_I2VS	SEL_I2VS
PRF_LBIAS		RB_OBVS	SAV_I3VL	
PRF_LERR		RB_RAMP	SAV_I3VS	
PRF_LFDBK		RB_RSAF	SAV_I4VL	
PRF_LMEA		RB_RTHI	SAV_I4VS	
PRF_MESP		RB_RTLO	SAV_I5VL	
PRF_MEVL		RB_RTVL	SAV_I5VS	
PRF_MEVS		RB_RTVS	SAV_NAVG	
PRF_MEZE			SAV_NCOL	
PRF_OTSP				
PRF_OTZE				
PRF_RAVL				
PRF_RAVS				
PRF_REVL				
PRF_REVS				
PRF_SPHI				
PRF_SPINT				
PRF_SPLO				
PRF_SPRT				
PRF_SPVL				
PRF_SPVS				
PRF_FDSP				
PRF_FDVL				

XBR
XBR_CPVL
XBR_HISPEC
XBR_LCPKV
XBR_LCPVL
XBR_LOSPEC
XBR_LRNGV
XBR_LXBRV
XBR_RLCL
XBR_RMAX
XBR_RMIN
XBR_RNGV
XBR_RTARG
XBR_RUCL
XBR_STDDEV
XBR_XBALM
XBR_XBRV
XBR_XLCL
XBR_XTARG
XBR_XUCL

Diagram of Hardware Connection



Chapter 2. Principles of Operation

At startup the interface loads all the PI points with the designated point source and interface number. As each point is loaded a call is made to the D/3 system to verify the existence of the point in the DBA. GLF calls are used for continuous points and get_vseq is used for sequence points.

The startup command line controls whether the Sequence Database is accessible or exists. If the interface should not attempt to read the Sequence Database and Sequence Database PI points should be rejected, the command line should include **`/noseq`**.

After the PI points are loaded, the interface periodically reads data from the D/3 or outputs data on event. The interface can request or send up to 255 points of the same D/3 database block type at the same time for collection of continuous data. There is an issue with this grouping, though: it is more efficient to group the requests, but the D/3 does not return error information point-by-point. This means that if the request fails all the tags would be considered **`BAD INPUT`**. The interface allows for two options when such an error occurs:

- The interface can go on to individually requesting data for each of the points and eventually determine the point or points in error. These points are then added to a list to revisit at a later time, about thirty minutes later. This waiting period allows the user to change points in the D/3 without having to cycle the interface. If on the second pass the GLF call still returns an error for the D/3 point, the PI point is removed from the interface. The interface attempts to add the PI point again when the interface is restarted or if a non-essential PI point attribute is changed.
- The user may want the interface to handle the GLF errors by stopping the interface automatically. In this case, use **`/KS`** or **`/KS=0`** as a command-line parameter.

UniInt Failover

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

Chapter 3. Installation Checklist

If you are familiar with running 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 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. 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***)**
7. If you will use digital points, define the appropriate digital state sets.
8. Add the On and Off states to the System Digital State Set.

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

   `Location1` specifies the Interface instance ID.
   `Location2` is the list number for continuous data and is also used to specify whether a point is input or output.
   `Location3` is used to specify the type and extent of data quality checking on Continuous Database items to be performed by the interface.
   `Location4` specifies the scan class.
   `Location5` is used to specify bit-masking operations for D/3 continuous database elements, short int data types only.
   `ExDesc` specifies whether the point is a Continuous Database point, a Sequence Database point, a Performance point, or a Trigger-based Input point.
   `InstrumentTag` specifies the D3 element (EPN) to be accessed.

10. Start the Interface interactively and confirm its successful connection to the PI Data Archive without buffering.
11. Confirm that the Interface collects data successfully.
12. Stop the Interface and configure a buffering application (either Bufserv or PIBufss). When configuring buffering use the ICU menu item Tools → Buffering… → Buffering Settings to make a change to the default value (32678) for the Primary and Secondary Memory Buffer Size (Bytes) to 2000000. This will optimize the throughput for buffering and is recommended by OSIsoft.
13. Start the buffering application and the Interface. Confirm that the Interface works together with the buffering application by either physically removing the connection between the Interface Node and the PI Data Archive Node or by stopping the PI Data Archive.
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 *UniInt Interface User Manual* 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 PI Interface Nodes instead of directly on the PI Data Archive node. A PI 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 PI 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 PI 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 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 PI, 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 *UniInt Interface User Manual* for special procedural information.

Naming Conventions and Requirements

In the installation procedure below, it is assumed that the name of the interface executable is `TID3.exe` and that the startup command file is called `TID3.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, `TID31.exe` and `TID31.bat` would typically be used for interface number 1, `TID32.exe` and `TID32.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.

Interface Installation Directory

The interface install kit will automatically install the interface to:

PIHOME\Interfaces\TID3\

PIHOME is defined in the pipc.ini file.

Interface Installation Procedure

The PI GSETID3 Interface setup program uses the services of the Microsoft Windows Installer. Windows Installer is a standard part of Windows 2000 and later operating systems. To install, run the appropriate installation kit.

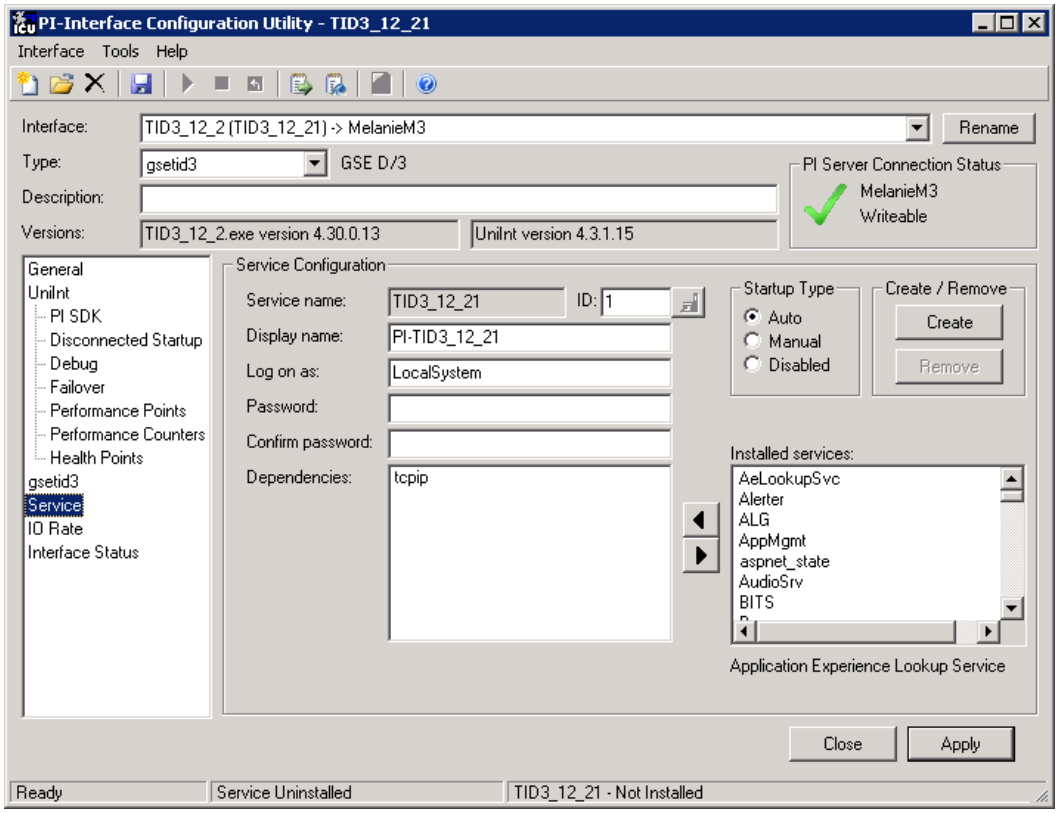
```
GSETID3_#.#.#.#_.exe
```

Installing Interface as a Windows Service

The PI GSETID3 Interface service can be created, preferably, with the PI Interface Configuration Utility, or can be created manually.

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:



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

Password

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

Confirm password

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

Dependencies

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

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

Note: Please see the 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*.

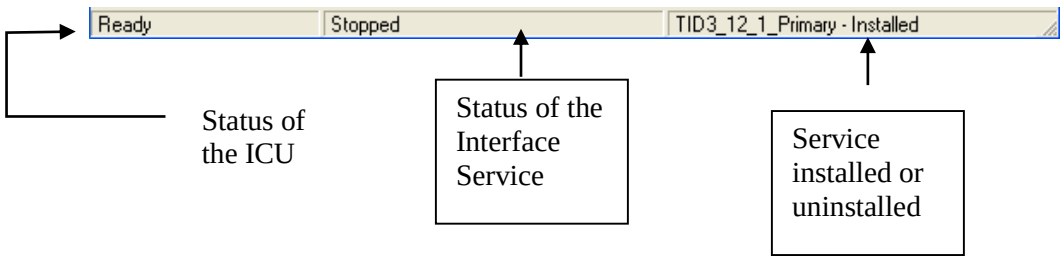
Remove

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.



Installing Interface Service Manually

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

```
TID3.exe /help
```

Open a Windows command prompt window and change to the directory where the `TID31.exe` executable is located. Then, consult the following table to determine the appropriate service installation command.

Windows Service Installation Commands on a PI Interface Node or a PI Data Archive Node with Bufserv implemented	
Manual service	`TID3.exe /install /depend "tcpip bufserv"`
Automatic service	`TID3.exe /install /auto /depend "tcpip bufserv"`
*Automatic service with service id	`TID3.exe /serviceid X /install /auto /depend "tcpip bufserv"`
Windows Service Installation Commands on a PI Interface Node or a PI Data Archive Node without Bufserv implemented	
Manual service	`TID3.exe /install /depend tcpip`
Automatic service	`TID3.exe /install /auto /depend tcpip`
*Automatic service with service id	`TID3.exe /serviceid X /install /auto /depend tcpip`

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

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

Chapter 5. 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 PI as digital state sets. Each digital state set is a user-defined list of strings, enumerated from 0 to n to represent different values of discrete data. For more information about PI digital tags and editing digital state sets, see the PI Data Archive manuals.

An interface point that contains discrete data can be stored in PI 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 PI instead of a value. The system digital state set has many unused states that can be used by the interface and other PI clients. Digital States 193-320 are reserved for OSIsoft applications.

Chapter 6. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For example, the string *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 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.

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 PI are associated with default PointSource characters. The Totalizer Subsystem uses the PointSource character `T`, the Alarm Subsystem uses @ for Alarm Tags, `G` for Group Alarms and `Q` for SQC Alarm Tags, 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. D/3 Point Configuration

D/3 Versions 5.0 to 9.0-1

Changes to the D/3 database causes the shared memory regions that keep track of the Ethernet addresses and the display databases to be modified, but they are not updated in the interface. Subsequent calls to the D/3 continuous database will return –69 errors. The interface will rebuild any list that returns a –69 error, deleting the tag with the offending IPN from the list. The PI tag will be set to BAD INPUT and internally set to scan off. Every 10 minutes for up to 40 minutes the interface will attempt to add the tag back into an appropriate GLF list. If it cannot be added back in, the tag will be deleted from the interface. If the D/3 point is added back into the database later, the PI point will need to be modified in order for the interface to add the tag again.

Note: It is paramount that all changes to the D/3 database be made from a single file to prevent cycling within the interface for every point change.

D/3 Versions 9.0-2 and Higher

Changes to the D/3 database causes the shared memory regions that keep track of the Ethernet addresses and the display databases to be modified, but they are not updated in the interface. Starting with version 9.0-2, NovaTech provides a mechanism to programmatically recognize that the D/3 database has changed. The interface will rebuild any list that returns a –69 error, deleting the tag with the offending IPN from the list. The PI tag will be set to BAD INPUT and internally set to scan off. Every 10 minutes for up to 40 minutes the interface will attempt to add the tag back into an appropriate GLF list. If it cannot be added back in, the tag will be deleted from the interface. If the D/3 point is added back into the database later, the PI point will need to be modified in order for the interface to add the tag again.

To activate this feature, use the **`/CH`** command-line parameter.

Note: It is paramount that all changes to the D/3 database be made from a single file to prevent cycling within the interface for every point change.

D/3 Point Lists

The interface can request or send up to 255 points of the same D/3 database block type at the same time for collection of continuous data. Several different field types can be included in a single request. This means that when PI points are grouped, it is important to be aware of the distribution of fields associated with the set of points within the group. If there are several PI points getting data from the same unit, block type, and the same set of fields, this will result in very efficient data access from the D/3 because all of the required data will be retrieved in one transaction. However, if a large group of PI tags accesses a large number of different blocks, but some points access many data fields and most access only a few fields, some inefficiency will result because the transaction that results will include all requested fields for all requested blocks.

To change the allocation of points to lists, edit the point attributes by using the POINT BLD display or by using PIConfig in PI 3. If massive changes are made to the PI point database, it is advisable to stop and restart the D/3 interface in order to expedite the changes taking effect.

Note: The most efficient list building is done by the interface and this is the recommended method of operation. Location2 should be set to 0 for the interface to build the lists.

Chapter 8. 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.

Point Attributes

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

Tag

The Tag attribute (or tagname) is the name for a point. There is a one-to-one correspondence between the name of a point and the point itself. Because of this relationship, 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 tags whose length is at most 255 or 1023 characters. The following table indicates the maximum length of this attribute for all the different combinations of PI API and 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 tags. Similarly, a floating-point value from the device can be sent to integer or digital PI tags, although the values will be truncated.

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

Numerical & Discrete

Typically the PI point type will match the D/3 field data type. It is possible to use a PI point type that does not match the D/3 point type since the interface determines the D/3 point type when retrieving the D/3 internal IDs. Note that D/3 integer values can range between -32767 and 32767 so OSIsoft recommends configuring a type floating-point PI point if negative integer values are expected.

Strings

This interface supports reading of strings from the NovaTech D/3 when the PI home node is PI 3. Configure the point the same as any other point. String input to PI 3 is supported from an interface running on Windows. String output to the NovaTech D/3 is only supported from Windows

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 is the list number for continuous data and is also used to specify whether a point is input or output. Since Sequence variables and outputs do not use list numbers, Location2 should be zero for Sequence variables that are inputs and a negative value should be used for variables that are outputs.

Outputs = negative number as described below

Sequence variables = 0

When collecting continuous data, the interface sends and receives values for a list of D/3 fields in each message.

Note: The most efficient list building is done by the interface and this is the recommended method of operation. Location2 should be set to 0 for the interface to build the lists.

- Continuous data – interface builds lists = 0

However, the user can control the grouping of fields into lists by specifying the list number in Location2. List numbers start at 1 and may be repeated for each scan class (Location 4), D/3 Unit, and D/3 Block Type combination. In general, the most efficient grouping will occur when points are grouped requesting the same field together into a given list.

- Continuous data – user specifies list = 1 – X

Outputs

The user has the option of controlling what data gets sent, or not, to the DCS in the event that the PI value is a digital state. In this case, digital state means any digital state for a float or integer tag or a digital state not in the normal range for a digital state tag.

- -1: Enter –1 to have the current value of the tag sent to the DCS, even if it is a digital state. (This is the action taken in interface version 3.19 and earlier.)
- -2: Enter –2 to have **no** value sent to PI when the PI value is a digital state.
- -3: Enter –3 to replace a digital state PI value with the value entered with the **`/val`** command-line parameter.

Location3

Location 3 is used to specify the type and extent of data quality checking on Continuous Database items to be performed by the interface. A value of 1 in this Location indicates that the scan status of the D/3 scan block is to be checked. If the block is not being scanned, a **`No Data`** digital state will be sent to PI. If this Location contains a value of 2, the preceding check is performed and the status of the ALMCT field is checked for indications of D/3 hardware problems. If any problem is detected, a **`Bad Input`** state is generated. A value of 0 in this location indicates that neither of these checks is to be performed.

- Do not perform any data quality checking = 0
- Check if D/3 scan block is being scanned = 1
- Check if D/3 scan block is being scanned and check ALMCT field = 2

Note: Only Continuous Database blocks AI, DEV, DIN, DOT, and DGR allow quality checking.

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.

Location5

This location is used to specify bit-masking operations for D/3 continuous database elements, short int data types only. The least significant bit is labeled bit 1 and the most significant bit is bit 16.

A four-digit code is used to specify a group of contiguous bits within an integer value to be masked and shifted. The low bit number is provided in the lower two digits and the high bit number is provided in the upper two digits. The lesser of the 2 numbers is used for shifting. For example, a value in location 5 of 1205 or 0512 would mask out bits 5 through 12 and shift the result four places (bits) to the right. For example if bits 5 and 12 are set, the value returned is 129.

A value of 0707 would mask bit 7 and shift 6 bits to the right.

Note: The above examples may seem confusing, but the interface does a conversion to count the least significant bit as zero.

Zero in this location means that no bit masking is to be used.

A value of –1 in this location indicates that the number of the first bit that is set is to be returned. The D/3 value is checked starting from the least significant bit. If no bits are set, a value of zero is returned. For example, if the D/3 value is 4, a value of 3 is returned because the binary representation is 0100. If the D/3 value is 24 a value of 4 is returned because the binary representation is 011000.

InstrumentTag

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.

This contains the specification of the D/3 data element to be accessed.

For continuous data, this consists of the D/3 External Point Name, block number, and field type each separated by a colon, ":". An example of a continuous database specification is:

```
T1001:0:AI_MEAS
```

Note that the D3-specified Control Block can be selected by using CB as the block number. This will cause the block specified in the AI block as the D/3 control block to be accessed. As such, `TI1001:0:AI_MEAS` and `TI1001:CB:AI_MEAS` are equivalent.

For sequence data, the specification consists of Unit Name and variable name separated by a colon ":". An example of a sequence database item specification is:

```
UNITX:variable1
```

> **Note:** The colon delimiter may be replaced by specifying a different character in the **`/itdelim=char`** parameter. For example, the EPN T1:003 cannot be read if the delimiter is a colon. If the delimiter is changed with **`/itdelim=|`**, then the InstrumentTag could be **`T1:003|0|AI_MEAS`**. The new delimiter will need to be used in every InstrumentTag when **`/itdelim=char`** is used.

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.

Continuous Database Point

For Continuous Database items the extended descriptor must contain: D3C

Sequence Database Point

For Sequence Database items the extended descriptor must contain: D3S

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 Tag 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 tag. 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 pipc.log and the tag 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 tag to be rejected by the Interface, SCAN OFF will be written to the PI point.

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 when PI 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 Server is down for maintenance, upgrades, backups, and unexpected failures. That is, when PI 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.

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.

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 tag is specified with the SourceTag attribute instead of with the "event" or "trig" keywords. Otherwise, the behavior of event conditions described under '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 tag to receive a digital state that is indicative of the failure, which is very important for troubleshooting.

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.

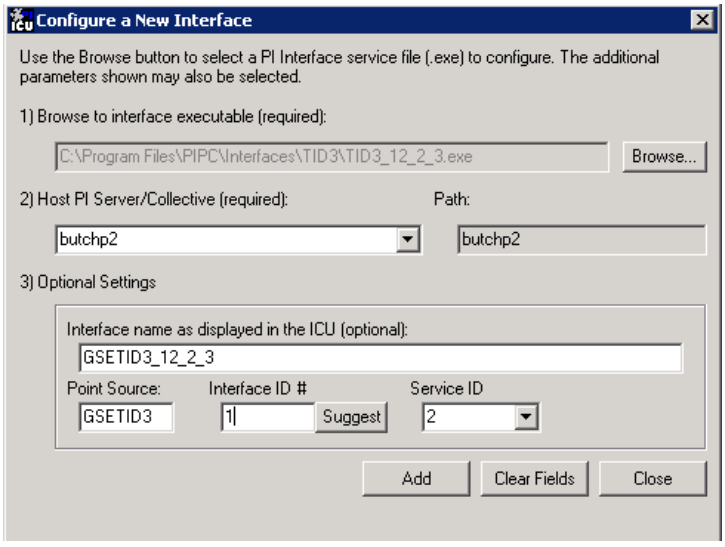
The PI Interface Configuration Utility (PI ICU) provides a tool for configuring the Interface startup command file.

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 (`TID3.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 GSETID3 Interface.

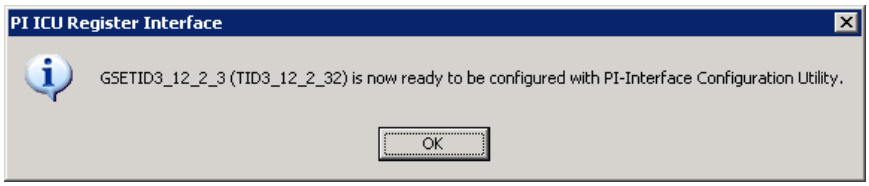
From the PI ICU menu, select *Interface*, then *NewWindows Interface Instance from EXE...*, and then *Browse* to the `TID3.exe` executable file. Then, enter values for *Point Source* and *Interface ID#*. A window such as the following results:



"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*.

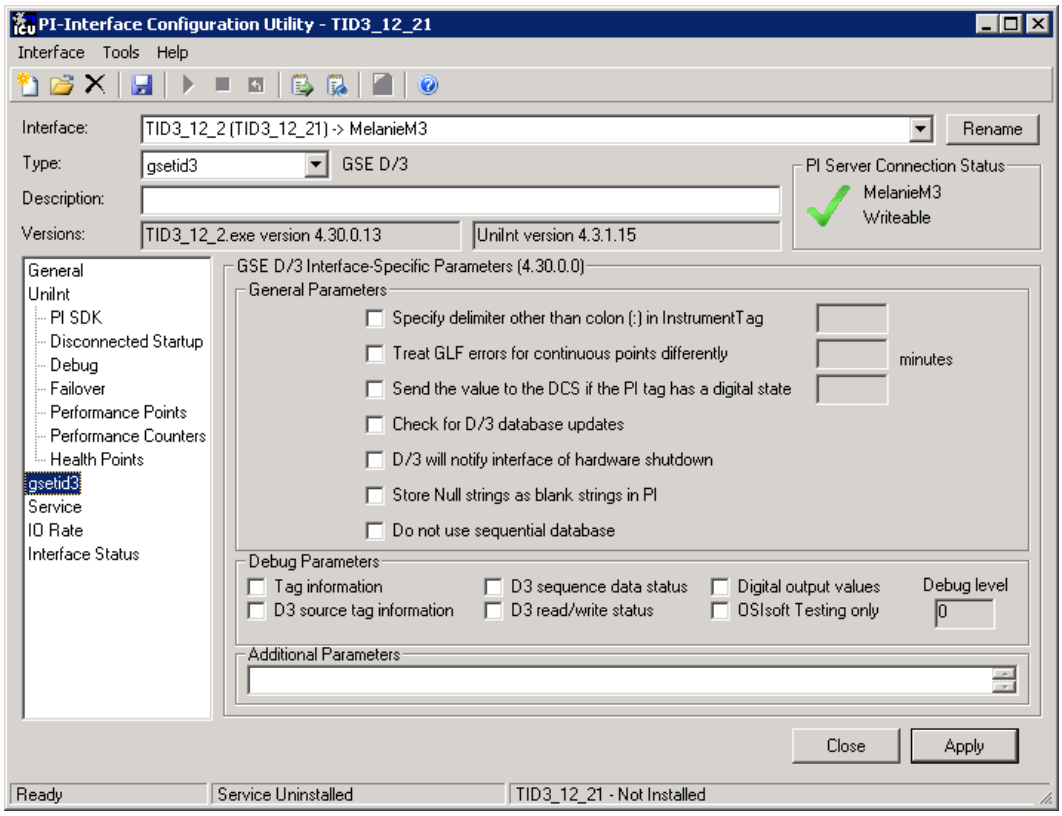
The following display should appear:



Note that in this example the Host PI System is butchp2. To configure the interface to communicate with a remote PI Data Archive, 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 `GSETID3`. If not, use the drop-down box to change the Interface *Type* to be `GSETID3`."

Click on Apply to enable the PI ICU to manage this copy of the PI GSETID3 Interface.



Since the GSETID3 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 *GSETID3* 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.

GSETID3 Interface page

Since the startup file of the PI GSETID3 Interface is maintained automatically by the PI ICU, use the *GSETID3* 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.

Specify delimiter other than colon ":" in InstrumentTag

This optional parameter is the character used as the delimiter in the InstrumentTag point attribute. By default, a colon ":" is used as the InstrumentTag delimiter. This is useful when the EPN on the D/3 uses embedded colons. Note that lower-case characters and the following list of characters are not allowed for /ITDELIM values:

% ^ & " | < >

The command-line parameter equivalent is (**/ITDELIM=x**).

Treat GLF errors for continuous points differently

This optional parameter indicates that the interface should treat GLF errors for continuous points differently. By default, GLF errors will cause the interface to rebuild the internal lists and recovery takes place. If this option is selected in the ICU Control, the user can specify the number of minutes to delay after a GLF error occurs before making another GLF call. **(/KS=#)**.

> **Note**: The /KS option is provided for only specific instances where it has been needed. In General, this option should NOT be used.

Send the value to the DCS if the PI tag has a digital state

This optional parameter allows the user to specify a value that will be sent to PI if the PI tag has a digital state. This value is used in conjunction with a value of -3 in the PI tag's Location2 attribute. **(/VAL=#)**

Check for D/3 database updates

This optional parameter notifies the interface to programmatically check for D/3 database updates. If this option is NOT specified, the interface will only detect changes when a -69 error is returned from a GLF call. This option should always be used on Windows. (**/CH**).

D/3 will notify interface of hardware shutdown

This optional parameter indicates that the D/3 can notify the interface that there is a hardware shutdown. The interface will shut down after writing shutdown messages to each input tag. This option should always be used on Windows. (**/HW**)

Store Null strings as blank strings in PI

This optional parameter will cause NULL strings to be stored in PI as blank strings. The default is to leave the string NULL. **(/BL)**

Do not use sequential database

This optional parameter tells the interface not to use the sequential database or that the sequential database is unavailable. **(/NOSEQ)**

Debug Parameters

For debugging purposes the user can set the interface to send extra interface specific messages to the pipc.log file. There are six different types of debugging messages. These will be sent to the log depending on which check box was selected in the Debug Parameters section of the GSETID3 tab. (**/DB=#**, where **#** is **1**, **2**, **4**, **8**, **16**, **32** or a combination of any or all values, maximum**=63)**

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

Parameter	Description
/bl Optional	Causes NULL strings to be stored in PI as blank strings. The default is to leave the string NULL.
/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)
/ch Recommended	Notifies the interface to programmatically check for D/3 database updates. If this option is not specified, the interface will only detect changes when a –69 error is returned from a GLF call. It is recommended this be used.

Parameter	Description
/db=# **/db** Optional	This is an optional command line switch that will turn on additional interface debugging messages. (#=message level.) Supported debug levels: **/db=1** D/3 tag information **/db=2** D/3 source tag information **/db=4** D/3 sequence data status **/db=8** D/3 read/write status **/db=16** Digital output values **/db=32** Internal testing use only **Note**: Debug levels may also be combined. For example, to print all messages from levels 1 and 2, set the debug level to 3 (**/db=3**).
/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, each copy of the interface that is running without **/ec=#**explicitly defined will write to the same I/O Rate point. This means either explicitly defining an event counter other than 1 for each copy of the interface or not associating any I/O Rate points with event counter 1. Configuration of I/O Rate points is discussed in the section called I/O Rate Point. 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. One must consult the interface-specific documentation to see whether subsequent instances of the **/ec** parameter have any effect. 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.
/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**

Parameter	Description

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

`/host=host:port` Required	The **`/host`** parameter is used to specify the PI Home node. `Host` is the IP address of the PI Sever node 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 a PI Interface Node, the domain name of the PI home node is Marvin, and the IP address of Marvin is 206.79.198.30. Valid **`/host`** parameters would be: **`/host=marvin`** **`/host=marvin:5450`** **`/host=206.79.198.30`** **`/host=206.79.198.30:5450`**

Parameter	Description
/hw Recommended	Indicates that the D/3 can notify the interface that there is a hardware shutdown. The interface will shut down after writing shutdown messages to each input tag. It is recommended to use this option.
/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**
/itdelim=x Default: **/itdelim=:**	Specifies a delimiter other than a colon for use in the InstrumentTag point attribute. This is useful when the EPN on the D/3 uses embedded colons. **Note**: The following characters are not allowed for the **/itdelim** value: % ^ & " | < > In addition, lower-case characters are also not supported in the **/itdelim** field.
/ks=# Optional	Indicates that the interface should treat GLF errors for continuous points differently. Normal operation is that GLF errors cause the interface to rebuild the internal lists and recovery takes place. If the parameter is **/KS** or **/KS=0**, the interface will exit immediately upon ANY error from the GLF call. If the parameter is **/KS=#** where # is a positive number, the interface will delay # number of minutes before making another GLF call. This allows time for maintenance of connection to the D/3. **Note**: There has been only one customer who has needed this option. OSIsoft recommends that you run the interface without it unless errors occur.
/maxstoptime=stoptime Optional	When a Windows service is stopped, the service control manager spawns a new thread for the exit handler. The exit handler sets the "keep going" parameter for the interface to false and then waits a maximum of stoptime seconds for the main thread to reach a safe exit point before the exit handler continues with its cleanup operations. By default, stoptime is 120 seconds. If stoptime seconds are exceeded, the exit handler will continue with its cleanup operations and then force the interface to exit.
/noseq Optional	Do not use the sequential database or it is unavailable.

Parameter	Description
/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.
/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 tag. Next, the interface writes this value to each output tag. In addition, whenever an individual output tag is edited while the interface is running, the interface will write the current Snapshot value to the edited output tag. 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 tag is edited. In other words, when the **/sio** parameter is specified, outputs will only be written when they are explicitly triggered.
/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 3 Server, `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`.

Parameter	Description
/UFO_ID=# Required for UniInt Interface Level 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 This interface does not support unsolicited input interface failover control tags and therefore will not use this command line parameter to control the update interval."
/UFO_OtherID=# Required for UniInt Interface Level 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 Interface Level 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 `pipc.log` file stating the requested failover type is not supported.
/val=# Optional	Use this optional parameter in conjunction with –3 in location 2 to send the number # to the DCS when the tag has a digital state PI value.

Sample TID3.bat File

The following is an example file:

```
REM=================================================================
REM
REM TID3.bat
REM
REM Sample startup file for the PI Interface for GSE D/3 DBA
REM
REM=================================================================
REM
REM OSIsoft strongly recommends using PI ICU to modify startup files.
REM
REM Sample command line
REM
    TID3.exe ^
     /id=1 ^
     /ps=TID3 ^
     /hw ^
     /ch ^
     /host=XXXXXX:5450 ^
     /f=00:00:10 ^
     /f=5 ^
     /f=1
REM
REM End of TID3.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 the UniInt interface level 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 PI 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 *PI Data Archive Reference Guide*.

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

Quick Overview

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

This interface supports UniInt Failover (Phase 1 and Phase 2: warm, cold).

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 tags (input and output tags for three tag 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 tags:

  (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 tag updates.
- All other configuration parameters for the two interfaces must be identical.

Synchronization through the Data Source (Phase 1)

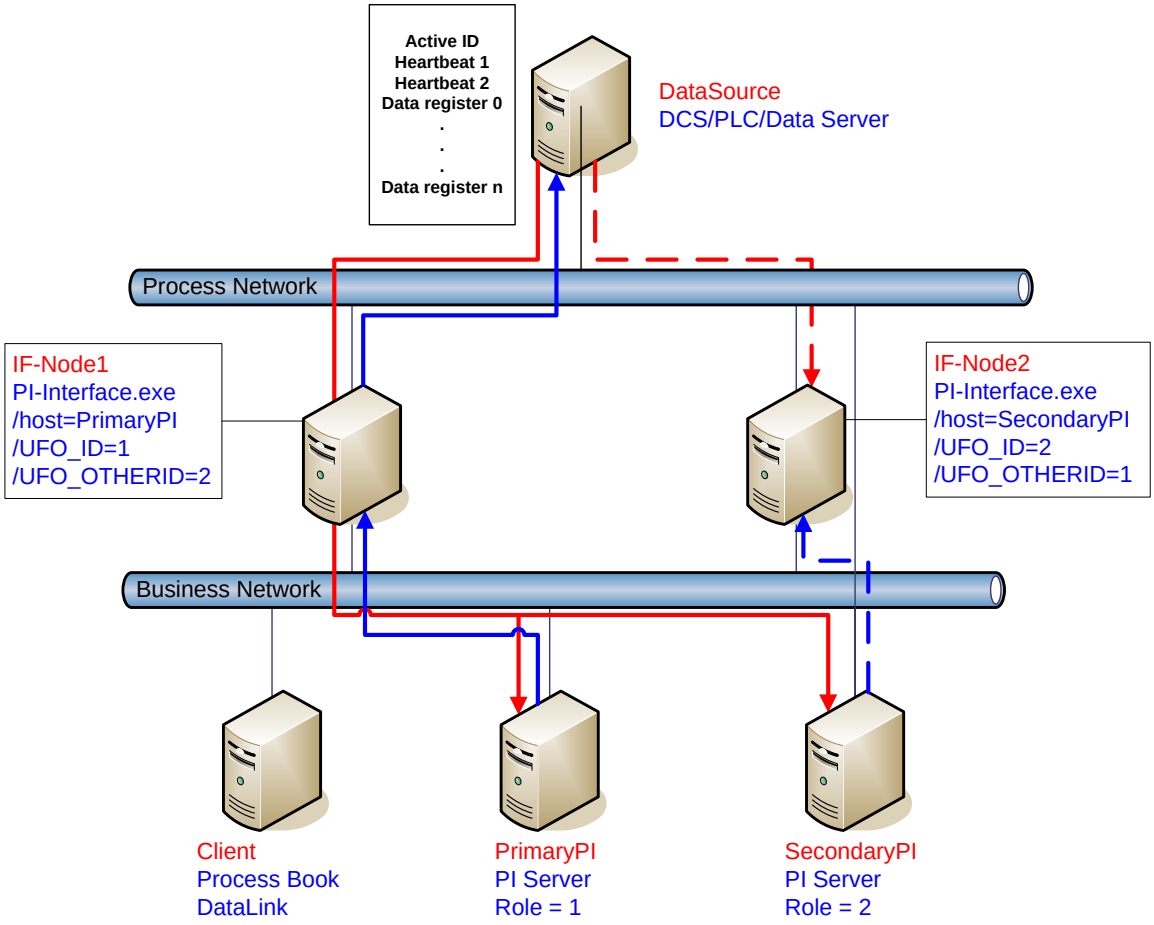


Figure 1: Failover Architecture Phase 1 - Synchronization through the Data Source

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 tags.

Configuring Synchronization through the Data Source (Phase 1)

<table>
<tr><th>Step</th><th>Description</th></tr>
<tr><td>1.</td><td>Verify non-failover interface operation as described in the Installation Checklist section of this manual</td></tr>
<tr><td>2.</td><td>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.
For Example: Using the Workbench supplied with the Cimplicity HMI Server, create and initialize the three required failover control points for the Cimplicity project - one Active ID and two Heartbeat points. See the Data Source Points section below.</td></tr>
<tr><td>3.</td><td>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 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 tags are unsolicited.</td></tr>
<tr><td>4.</td><td>Configure the PI tags
You must configure six PI tags for the interface (input and output tags for each of the three failover points on the data source). For more information about configuring input and output points, refer to the Interface Manual.
You can also configure two state tags for monitoring the status of the interfaces.
<table>
<tr><td>Tag</td><td>ExDesc</td><td>digitalset</td><td rowspan="9">The remaining attributes must be configured according to the interface manual so the tags map to the correct points on the data source</td></tr>
<tr><td>ActiveID_In</td><td>[UFO_ACTIVEID]</td><td></td></tr>
<tr><td>ActiveID_Out</td><td>[UFO_ACTIVEID]</td><td></td></tr>
<tr><td>IF1_HB_In
(IF-Node1)</td><td>[UFO_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF1_HB_Out
(IF-Node1)</td><td>[UFO_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF2_HB_In
(IF-Node2)</td><td>[UFO_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF2_HB_Out
(IF-Node2)</td><td>[UFO_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF1_State
(IF-Node1)</td><td>[UFO_STATE:#]</td><td>IF_State</td></tr>
<tr><td>IF2_State
(IF-Node2)</td><td>[UFO_STATE:#]</td><td>IF_State</td></tr>
</table></td></tr>
<tr><td>5.</td><td>If using PI APS to synchronize the Data Source and PI points, special attention must be paid to the failover control points and tags. Check that the failover control points and tags are not included in the PI APS synchronization scheme. Synchronizing the control points will cause the failover tags to be edited by PI APS and may result in possible interface shutdown.</td></tr>
<tr><td>6.</td><td>Test the configuration.</td></tr>
</table>

Step	Description
	Run the interface with the six Failover Control PI tags to ensure their proper operation. 1. Start the primary interface interactively without buffering. 2. Verify a successful interface start by reviewing the `pipc.log` file. The log file will contain messages that indicate the failover state of the interface. A successful start with only a single interface copy running will be indicated by an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 NovaTech D3 Server using the NovaTech D3 Point Control Panel. • The Active ID control point on the NovaTech D3 DBA Server must be set to the value of the running copy of the interface as defined by the **`/UFO_ID`** startup command-line parameter. • The Heartbeat control point on the NovaTech D3 Server 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 tag 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 tag 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 `pipc.log` file for the copy of the interface that was started first. The log file will contain messages that indicate the failover state of the interface. The state of this interface must have changed as indicated with an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to 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.

Step	Description
	• Review the `pipc.log` file for the copy of the interface that was started last. The log file will contain messages that indicate the failover state of the interface. A successful start of the interface will be indicated by an informational message stating "`UniInt failover: Interface in the "Backup" state.`" If the interface has failed to start, an error message will appear in the log file. For details relating to informational and error messages, refer to the Messages section below. 13. Verify data on the Novatech D3 Server using the NovaTech D3 Console. • The Active ID control point on the NovaTech D3 Server 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 NovaTech D3 Server 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 tag 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 tags 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 `pipc.log` file for a message indicating the backup interface has changed to the "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 PI. 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 `pipc.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 `pipc.log` file. 20. Test failover with different failure scenarios (e.g. loss of PI 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 PI 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.

Step	Description
	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 interface does not support unsolicited input interface failover control tags and therefore will not use this command line parameter to control the update interval.	50 - 20000 / **1000**
/Host=server	Required	Host PI Data Archive for Exceptions and PI tag 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 PI. **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 Tags

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

Active_ID Tag Configuration

Attributes	ActiveID IN	AcitveID 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 Tag 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 Tag 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 tags in more detail.

PI Tag ExDesc	Required / Optional	Description	Value / Default
[UFO_ACTIVEID] (Used for both the ActiveID IN and OUT tags.)	Required	The Active ID Input Tag must be configured as an input PI tag for the interface and it must be configured to read the **`ActiveID`** on the data source. Consult the *Interface User's Manual* for a description of configuring input tags. 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 Tag must be configured as an output PI tag for the interface and it must be configured to write to the Heartbeat 1 Point on the Data Source. Consult the Interface User Manual for a information about configuring output tags. 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 Tag must be configured as an input PI tag for the interface and it must be configured to read the Heartbeat 2 Point on the Data Source. Consult the Interface User Manual for a information about configuring input tags. 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 Tag ExDesc	Required / Optional	Description	Value / Default
[UFO_STATE:#] (IF-Node1)	Optional	The failover state tags are optional and do not require a point on the Data Source. The value of the state tag can be written to the Data Source by configuring a normal interface output tag and setting the SourceTag attribute to the failover state tag. The number following the colon (:) must be the Failover ID for the interface running on IF-Node1. The failover state tags are digital tags assigned to a digital state set with the following values. 0 = Off: The interface has been shut down. 1 = Backup No Data Source: The 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 PI.	0 - 5 / None Updated by the Primary Interface
[UFO_STATE:#] (IF-Node2)	Optional	The failover state tags are optional and do not require a point on the Data Source. The value of the state tag can be written to the Data Source by configuring a normal interface output tag and setting the SourceTag attribute to the failover state tag. 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

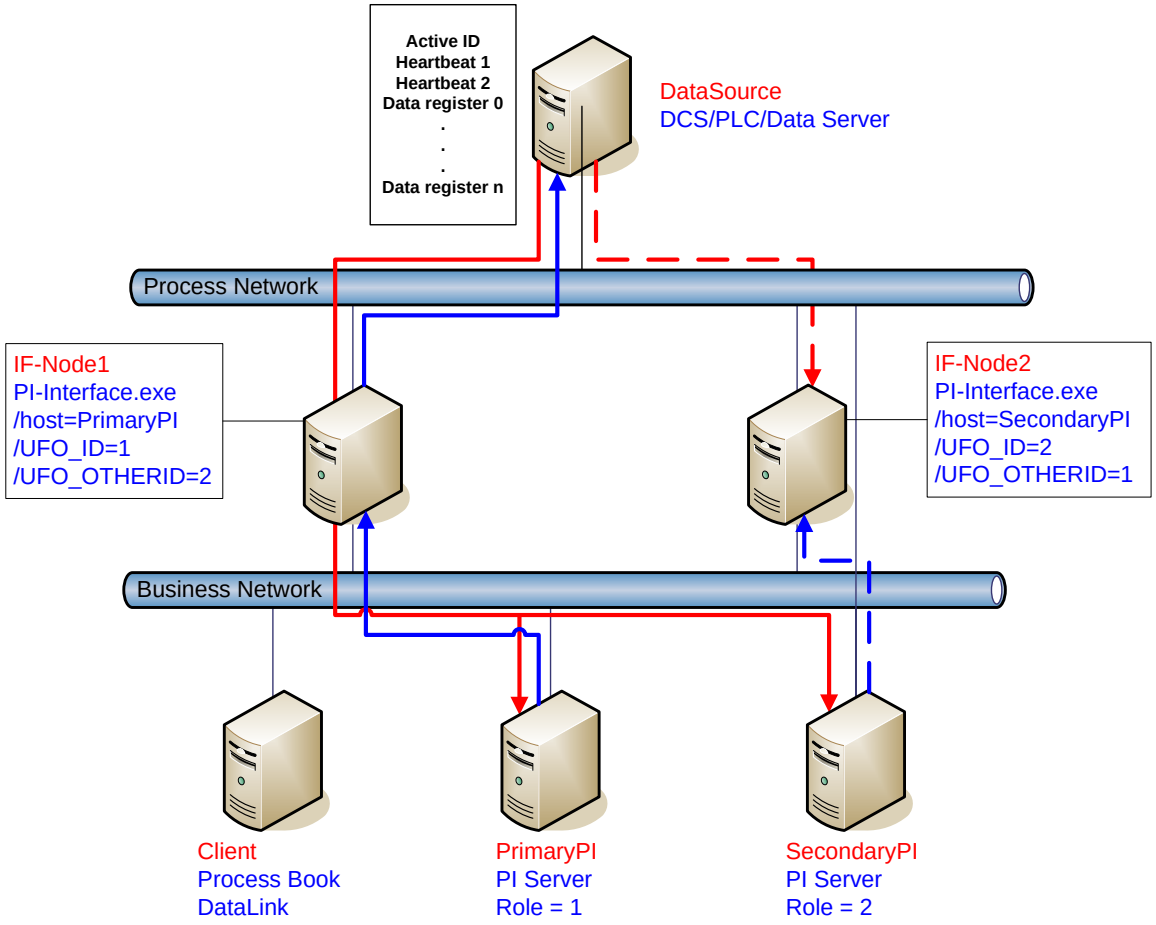


Figure 2 Synchronization through Data Source (Phase 1) Failover Architecture

Synchronization through the data source uses two separate interface nodes communicating with the data source. The failover scheme requires six tags on the PI Data Archive and three control points on the data source to control failover operation. The PI tags 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 tags is not required for failover operation because only the control points on the data source are monitored. However, the PI tag 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 **ActiveID** 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 PI and written to the data source) comes into PI 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 PI. 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 PI. 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 tags on the PI Data Archive. Updating of the control tags 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 BufSS in this case.

Each interface participating in the failover solution will queue 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 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 - 15 range. During a normal shutdown process, the heartbeat value will be 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)

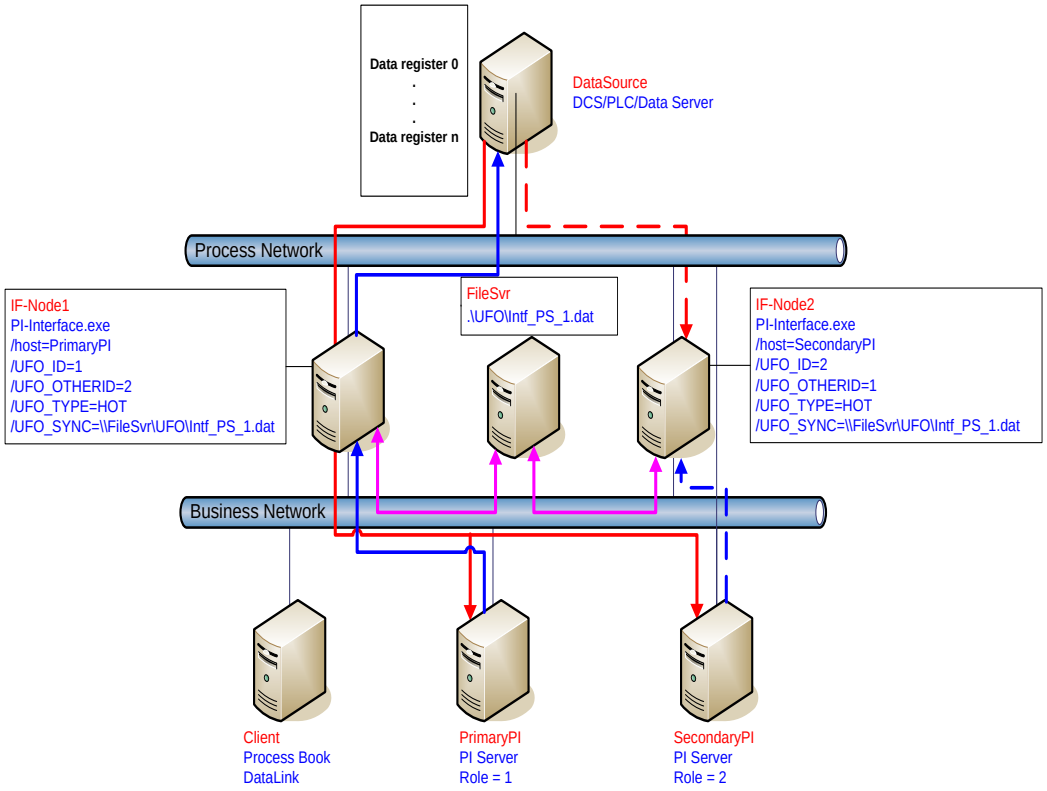


Figure 3: Synchronization through a Shared File (Phase 2) Failover Architecture

The Phase 2 failover architecture is shown in the figure above, 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)

<table>
<tr><th>Step</th><th>Description</th></tr>
<tr><td>1.</td><td>Verify non-failover interface operation as described in the Installation Checklist section of this manual</td></tr>
<tr><td>2.</td><td>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.</td></tr>
<tr><td>3.</td><td>Configure the interface parameters
Use the Failover section of the Interface Configuration Utility (ICU) to enable failover and create two parameters for each interface: (1) a Failover ID number for the interface; and (2) the Failover ID number for its backup interface.
The Failover ID for each interface must be unique and each interface must know the Failover ID of its backup interface.
If the interface can perform using either Phase 1 or Phase 2 pick the Phase 2 radio button in the ICU.
Select the synchronization File Path and File to use for Failover.
Select the type of failover required (Cold, Warm, Hot). The choice depends on what types of failover the interface supports.
Ensure that the user name assigned in the “Log on as:” parameter in the Service section of the ICU is a user that has read/write access to the folder where the shared file will reside.
All other command line parameters for the primary and secondary interfaces must be identical.
If you use a PI 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.</td></tr>
<tr><td>4.</td><td>Configure the PI tags
Configure five PI tags for the interface: the Active ID, Heartbeat 1, Heartbeat2, Device Status 1 and Device Status 2. You can also configure two state tags for monitoring the status of the interfaces.
Do not confuse the failover Device status tags with the UniInt Health Device Status tags. The information in the two tags is similar, but the failover device status tags are integer values and the health device status tags are string values.
<table>
<tr><th>Tag</th><th>ExDesc</th><th>digitalset</th><th rowspan="8">UniInt does not examine the remaining attributes, but the pointsource and location1 must match</th></tr>
<tr><td>ActiveID</td><td>[UFO2_ACTIVEID]</td><td></td></tr>
<tr><td>IF1_Heartbeat
(IF-Node1)</td><td>[UFO2_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF2_Heartbeat
(IF-Node2)</td><td>[UFO2_HEARTBEAT:#]</td><td></td></tr>
<tr><td>IF1_DeviceStatus
(IF-Node1)</td><td>[UFO2_DEVICESTAT:#]</td><td></td></tr>
<tr><td>IF2_DeviceStatus
(IF-Node2)</td><td>[UFO2_DEVICESTAT:#]</td><td></td></tr>
<tr><td>IF1_State
(IF-Node1)</td><td>[UFO2_STATE:#]</td><td>IF_State</td></tr>
<tr><td>IF2_State
(IF-Node2)</td><td>[UFO2_STATE:#]</td><td>IF_State</td></tr>
</table></td></tr>
<tr><td>5.</td><td>Test the configuration.</td></tr>
</table>

Step	Description
	After configuring the shared file and the interface and PI tags, the interface should be ready to run. See Troubleshooting UniInt Failover for help resolving Failover issues. 1. Start the primary interface interactively without buffering. 2. Verify a successful interface start by reviewing the `pipc.log` file. The log file will contain messages that indicate the failover state of the interface. A successful start with only a single interface copy running will be indicated by an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 tag 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 tag 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, 3, and 4. 7. Stop the backup interface. 8. Start buffering. 9. Start the primary interface interactively. 10. Once the primary interface has successfully started and is collecting data, start the backup interface interactively. 11. Verify that both copies of the interface are running in a failover configuration. • Review the `pipc.log` file for the copy of the interface that was started first. The log file will contain messages that indicate the failover state of the interface. The state of this interface must have changed as indicated with an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 `pipc.log` file for the copy of the interface that was started last. The log file will contain messages that indicate the failover state of the interface. A successful start of the interface will be indicated by an informational message stating "`UniInt failover: Interface in the "Backup" state.`" If the interface has failed to start, an error message will appear in the log file. For details relating to informational and error messages, refer to the Messages section below. 12. Verify data on the PI Data Archive using available PI tools. • The **Active ID** control tag 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.

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

Configuring UniInt Failover through a Shared File (Phase 2)

Start-Up Parameters

Note: The **/stopstat** parameter is disabled If the interface is running in a UniInt failover configuration. Therefore, the digital state, digstate, will not be written to each 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 File Synchronization Filepath and Optional Filename specify the path to the shared file used for failover synchronization and an optional filename used to specify a user defined filename in lieu of the default filename. The *path* to the shared file directory can be a fully qualified machine name and directory, a mapped drive letter, or a local path if the shared file is on one of the interface nodes. The *path* must be terminated by a slash (/) or backslash (\ ) character. If no terminating slash is found, in the **/UFO_Sync** parameter, the interface interprets the final character string as an optional *filename*. The optional *filename* can be any valid filename. If the file does not exist, the first interface	Any valid pathname / any valid filename The default filename is generated as *executablename_ pointsource_ interfaceID.dat*

Parameter	Required/ Optional	Description	Value/Default
		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 shut down and log an error to the `pipc.log` file stating the requested failover type is not supported.	COLD|WARM|HOT / **COLD**
`/UFO_Interval=#`	Optional	Failover Update Interval Specifies the heartbeat Update Interval in milliseconds and must be the same on both interface computers. This is the rate at which UniInt updates the Failover Heartbeat tags as well as how often UniInt checks on the status of the other copy of the interface.	50 - 20000 / **5000**

Parameter	Required/ Optional	Description	Value/Default
/Host=server	Required	Host PI Data Archive for Exceptions and PI tag 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 PI. **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 Tags

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

Active_ID Tag 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 Tag Configuration

Attribute	Heartbeat 1	Heartbeat 2	DeviceStatus 1	DeviceStatus 2
Tag	<HB1>	<HB2>	<DS1>	<DS2>
ExDesc	[UFO2_Heartbeat:#] Match # in **/UFO_ID=#**	[UFO2_Heartbeat:#] Match # in **/UFO_OtherID=#**	[UFO2_DeviceStat:#] Match # in **/UFO_ID=#**	[UFO2_DeviceStat:#] Match # in **/UFO_OtherID=#**
Location1	Match # in **/id=#**	Match # in **/id=#**	Match # in **/id=#**	Match # in **/id=#**
Location5	Optional, Time in 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 Tag 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 tags in more detail.

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

PI Tag ExDesc	Required / Optional	Description	Value
[UFO2_DEVICESTAT :#] (IF-Node1)	Required	Device Status 1 Tag The ExDesc must start with the case sensitive string: [UFO2_DEVICESTAT:#] The value following the colon (:) must be the Failover ID for the interface running on IF-Node1 The pointsource must match the interfaces' point source. Location1 must match the id for the interfaces. A lower value is a better status and the interface with the lower status will attempt to become the primary interface. The failover 1 device status tag is very similar to the UniInt Health Device Status tag except the data written to this tag are integer values. A value of 0 is good and a value of 99 is OFF. Any value between these two extremes may result in a failover. The interface client code updates these values when the health device status tag is updated.	0 - 99 / **None** Updated by the Interface on IF-Node1
[UFO2_DEVICESTAT :#] (IF-Node2)	Required	Device Status 2 Tag The ExDesc must start with the case sensitive string: [UFO2_DEVICESTAT:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2 The pointsource must match the interfaces' point source. Location1 must match the ID for the interfaces. A lower value is a better status and the interface with the lower status will attempt to become the primary interface.	0 - 99 / **None** Updated by the Interface on IF-Node2
[UFO2_STATE:#] (IF-Node1)	Optional	State 1 Tag The ExDesc must start with the case sensitive string: [UFO2_STATE:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node1 The failover state tag is recommended. The failover state tags are digital tags assigned to a digital state set with the following values. 0 = Off: The interface has been shut down.	0 - 5 / **None** Normally updated by the Interface currently in the primary role.

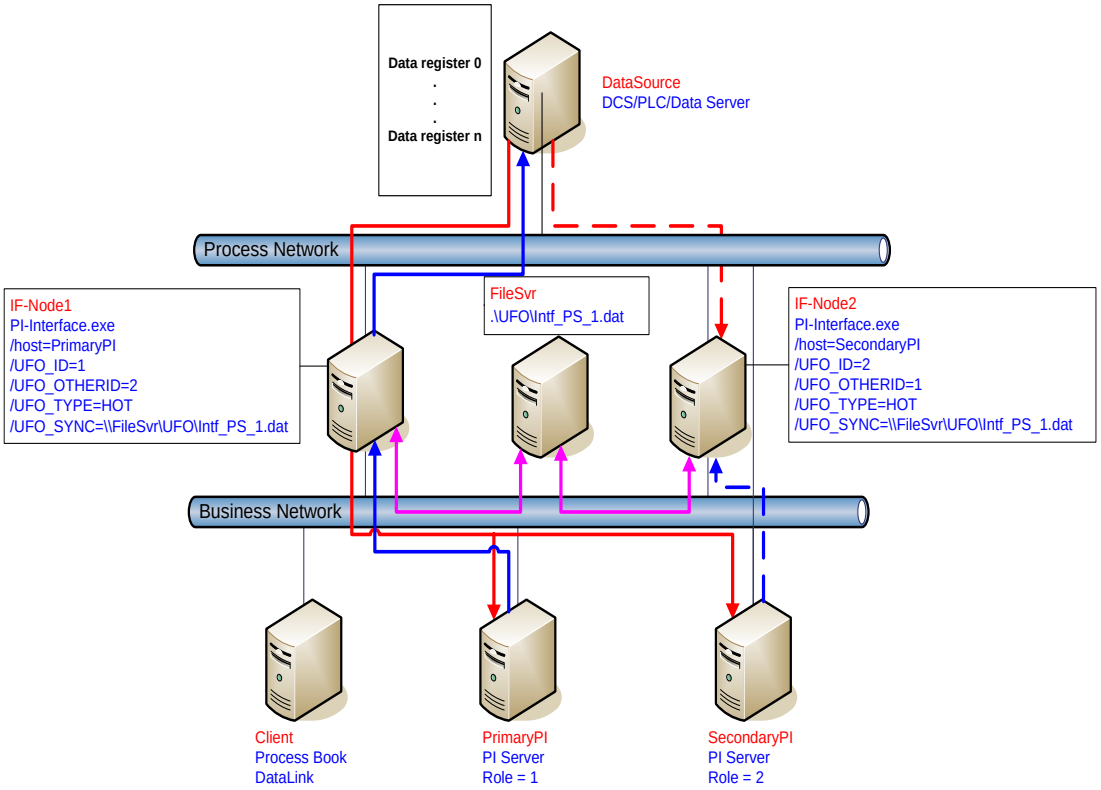
PI Tag ExDesc	Required / Optional	Description	Value
		1 = Backup No Data Source: The interface is running but cannot communicate with the data source. 2 = Backup No PI Connection: The interface is running and connected to the data source but has lost its communication to the PI Data Archive. 3 = Backup: 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 PI.	
[UFO2_STATE:#] (IF-Node2)	Optional	State 2 Tag The ExDesc must start with the case sensitive string: [UFO2_STATE:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node2 The failover state tag is recommended.	Normally updated by the Interface currently in the Primary state. Values range between 0 and 5. See description of State 1 tag.

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

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

Once the interface is configured and running, the ability to read or write to the PI tags is not required for the proper operation of failover. 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 tag 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 **ActiveID** on the data source to the backup failover ID.



The figure above shows a typical network setup in the normal or steady state. The solid magenta lines show the data path from the interface nodes to the shared file used for failover synchronization. The shared file can be located anywhere in the network as long as both interface nodes can read, write, and create the necessary file on the shared file machine. 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. The primary interface is also updating its heartbeat value; monitoring the heartbeat value for the backup interface, checking the active ID value, and checking the device status for the backup interface every failover update interval on the shared file. Likewise, the backup interface is updating its heartbeat value; monitoring the heartbeat value for the primary interface, checking the active ID value, and checking the device status for the primary interface every failover update interval on the shared file. As long as the heartbeat value for the primary interface indicates that it is operating properly, the **ActiveID** has not changed, and the device status on the primary interface is good, the backup interface will continue in this mode of operation.

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

The interaction between the interface and the shared file is fundamental to failover. The discussion that follows only refers to the data written to the shared file. However, every value written to the shared file is echoed to the tags on the PI Data Archive. Updating of the tags on 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 BufSS 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 - 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

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

As previously mentioned, in a hot failover configuration the backup interface actively collects data but does not send its data to PI. 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 PI. This queued data is sent to PI using the same function calls that would have been used had the interface been in a primary state when the function call was received from UniInt. If UniInt receives data without a timestamp, the primary copy uses the current PI time to timestamp data sent to PI. 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 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.

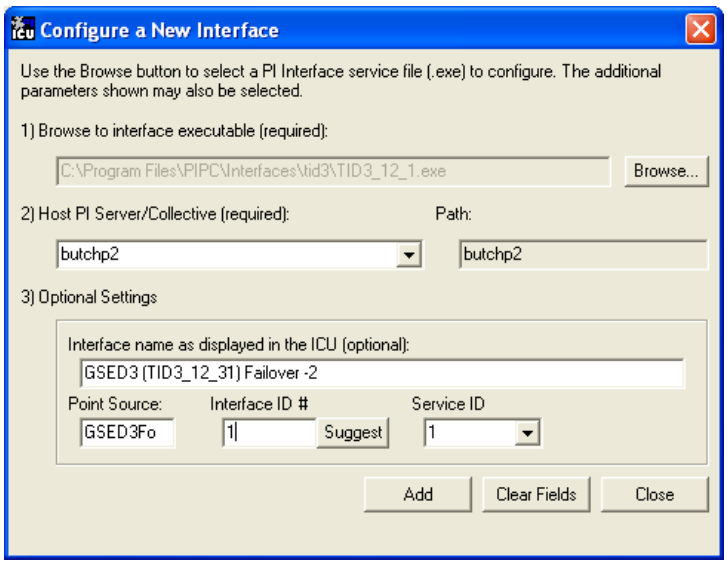


Figure 4: PI ICU configuration screen shows that the "PS/ID combo is already in use by the interface." The user must ignore the yellow boxes, which indicate errors, and click the *Add* button to configure the interface for failover.

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 2)section and Start-Up

Parameters for Phase 2, and in Configuring UniInt Failover through the Data Source (Phase 1) and Start-Up Parameters for Phase 1.

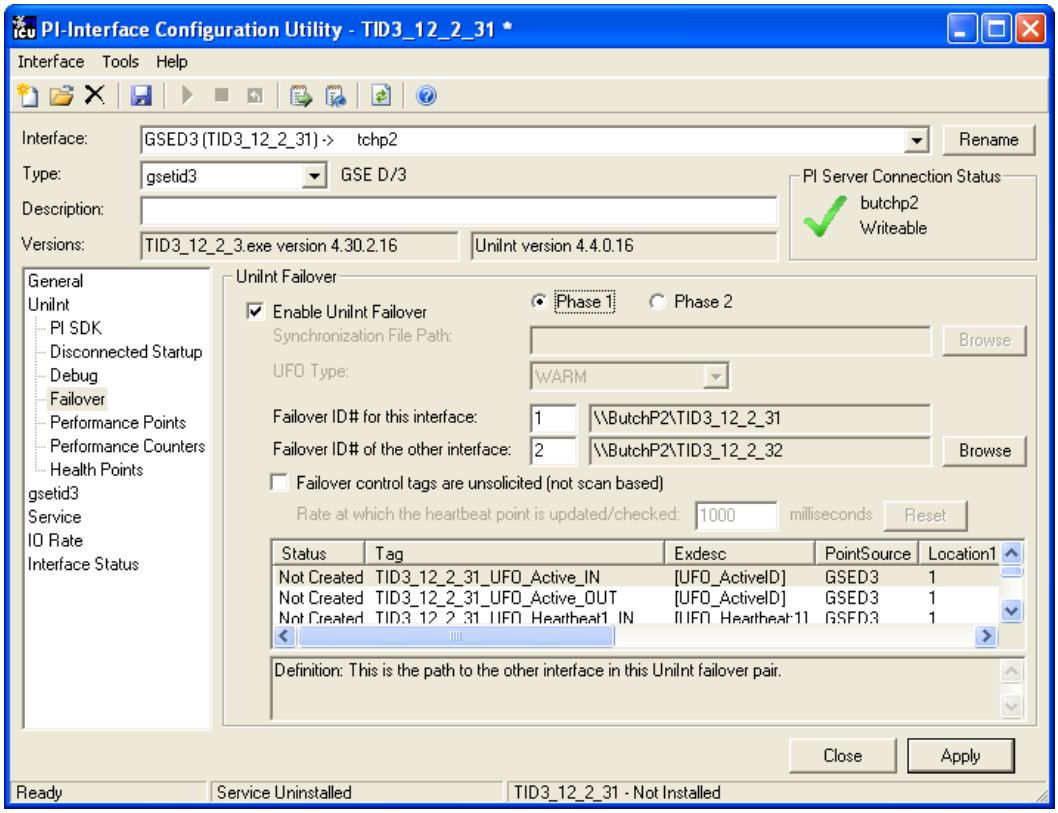


Figure 5: The figure above illustrates the PI ICU failover configuration screen showing the UniInt failover startup parameters (Phase 1). This copy of the interface defines its Failover ID as 2 (**`/UFO_ID=2`**) and the other interfaces Failover ID as 1 (**`/UFO_OtherID=1`**). The other failover interface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

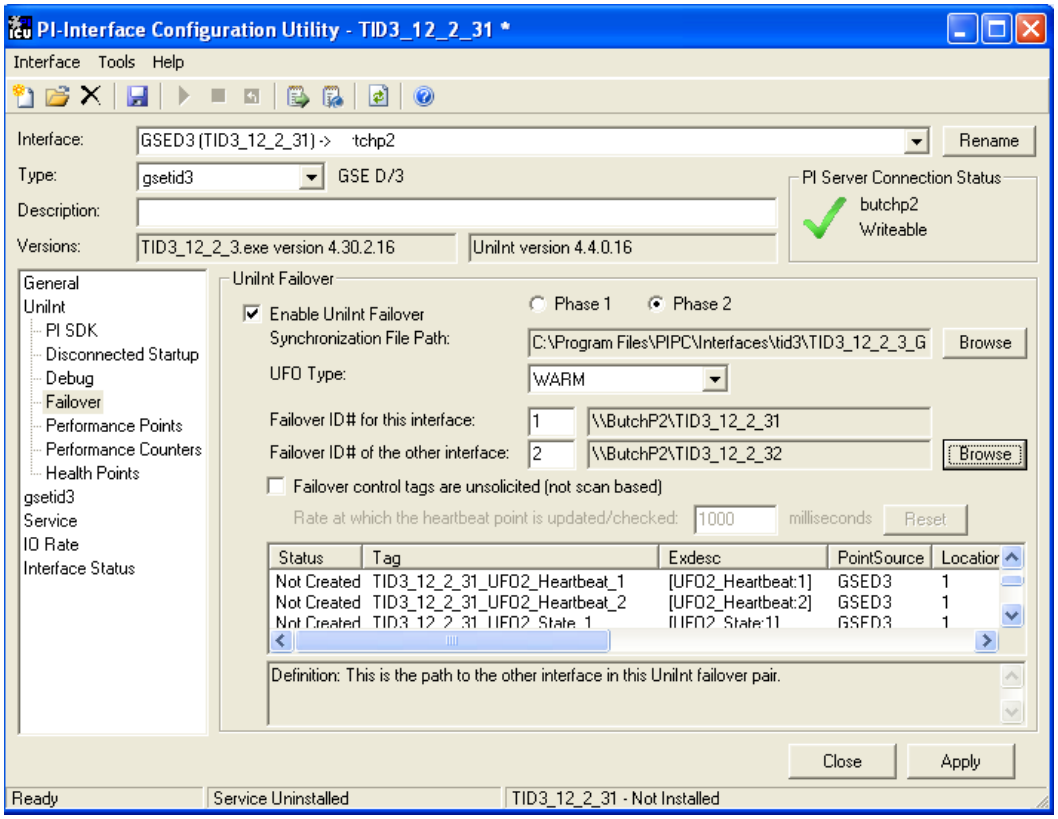


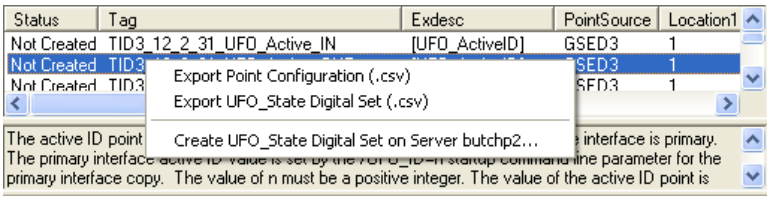
Figure 6: The figure above illustrates the PI ICU failover configuration screen showing the UniInt failover startup parameters (Phase 2). This copy of the interface defines its Failover ID as 2 (**`/UFO_ID=2`**) and the other interfaces Failover ID as 1 (**`/UFO_OtherID=1`**). The other failover interface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

Creating the Failover State Digital State Set

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

Using the 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 tags in the tag list and then select the "Create UFO_State Digital Set on Server XXXXXX…", where XXXXXX is the PI Data Archive where the points will be or are create on.



This choice will be grayed out if the UFO_State digital state set is already created 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) can be selected to create a comma separated file to be imported via the System Manangement Tools (SMT3) (version 3.0.0.7 or higher) or use the `UniInt_Failover_DigitalSet_UFO_State.csv` file included in the installation kit.

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

1. Open the SMT3 application.
2. Select the appropriate 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, select Points then 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" option. Refer to Figure 7 below.

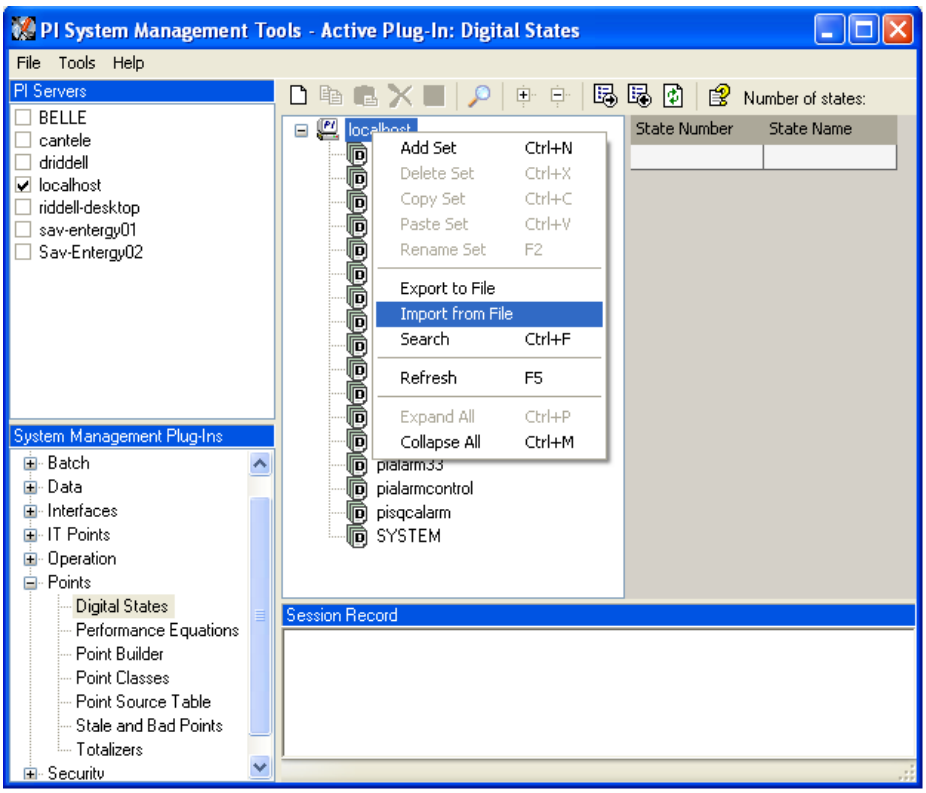


Figure 7: PI SMT application configured to import a digital state set file. The PI Servers window shows the "localhost" PI Data Archive selected along with the System Management Plug-Ins window showing the Digital States Plug-In as being selected. The digital state set file can now be imported by selecting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

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

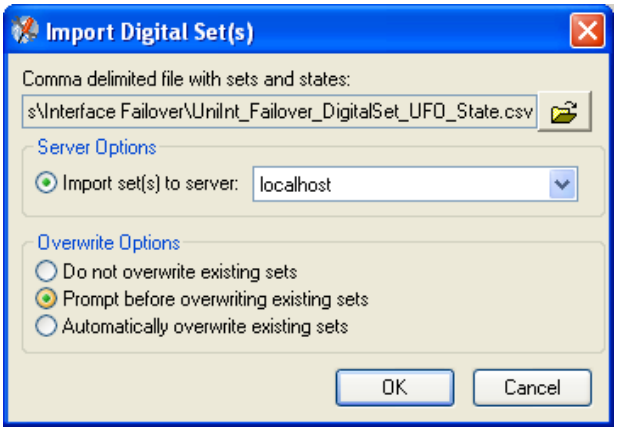


Figure 8: PI SMT application Import Digital Set(s) window. This view shows the `UniInt_Failover_DigitalSet_UFO_State.csv` file as being selected for import. Select the desired Overwrite Options by choosing the appropriate radio button.

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

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

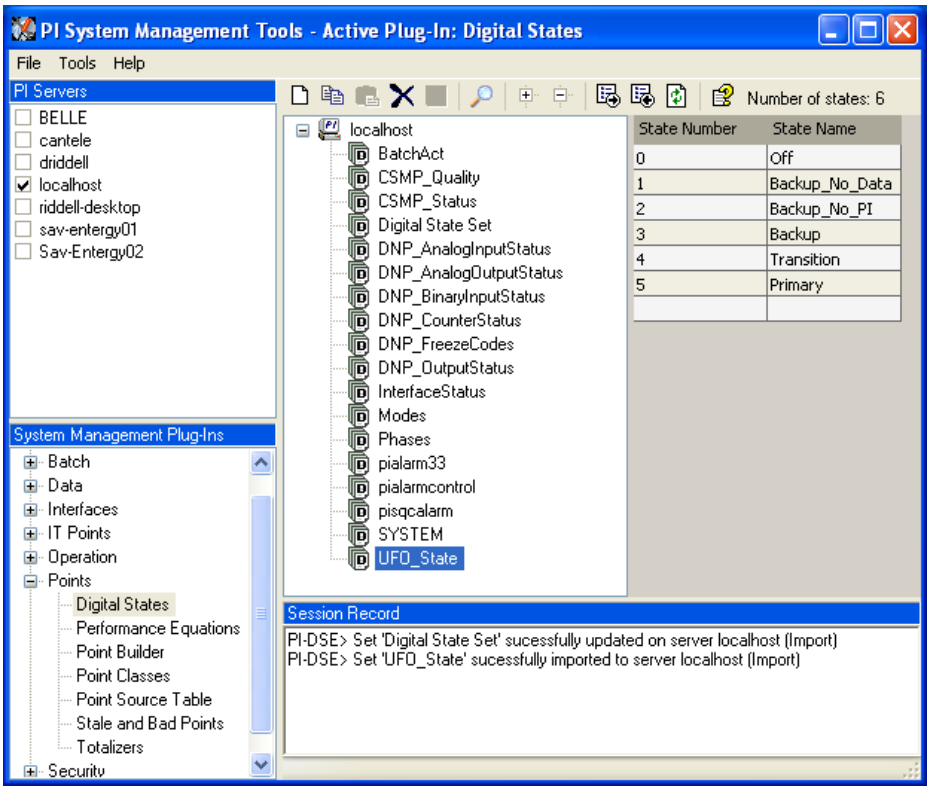


Figure 9: The PI SMT application showing the `UFO_State` digital set created on the "localhost" PI Data Archive.

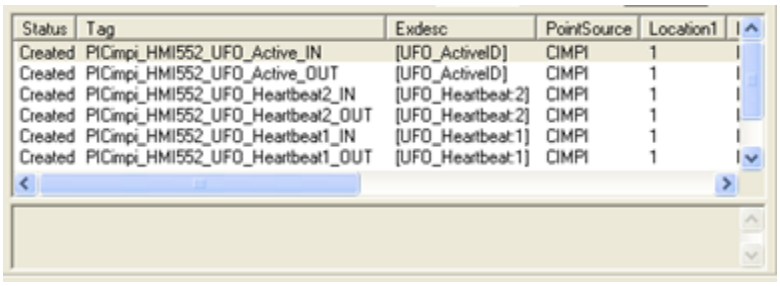
Creating the UniInt Failover Control and Failover State Tags (Phase 1)

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

In addition, the interface installation procedure installs an example file that already has the GSETID3 Interface specific attributes configured.

To use the ICU Failover page to create this file simply right click any of the failover tags in the tag list and select "Export Point Configuration" then edit this file as needed and import with SMT 3.

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



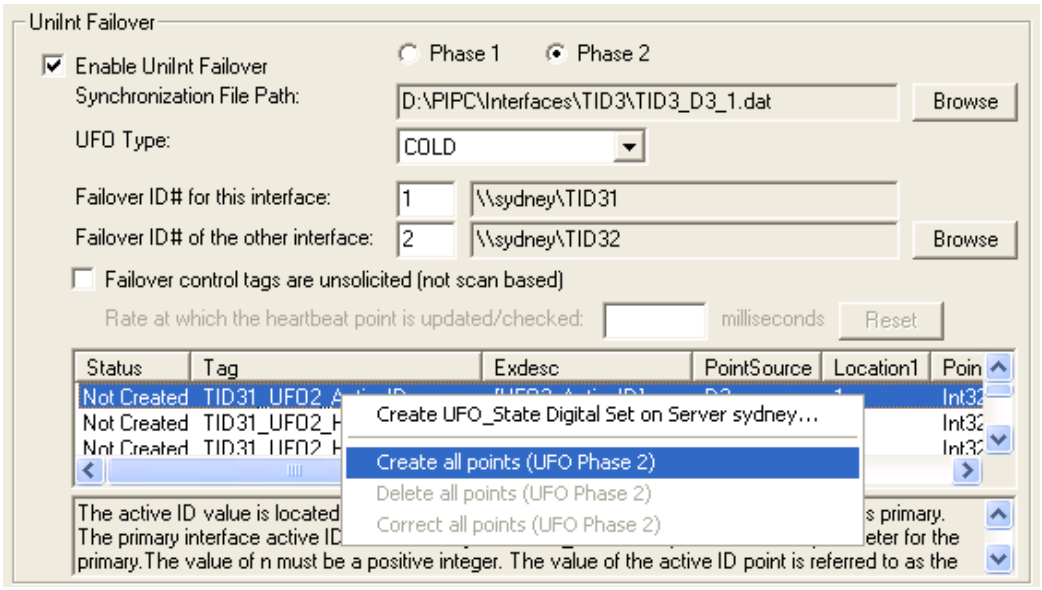
Status	Tag	Exdesc	PointSource	Location1
Created	PIClmpi_HMI552_UFO_Active_IN	[UFO_ActiveID]	CIMPI	1
Created	PIClmpi_HMI552_UFO_Active_OUT	[UFO_ActiveID]	CIMPI	1
Created	PIClmpi_HMI552_UFO_Heartbeat2_IN	[UFO_Heartbeat:2]	CIMPI	1
Created	PIClmpi_HMI552_UFO_Heartbeat2_OUT	[UFO_Heartbeat:2]	CIMPI	1
Created	PIClmpi_HMI552_UFO_Heartbeat1_IN	[UFO_Heartbeat:1]	CIMPI	1
Created	PIClmpi_HMI552_UFO_Heartbeat1_OUT	[UFO_Heartbeat:1]	CIMPI	1

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

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

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

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



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

Converting from Phase 1 to Phase 2 Failover

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

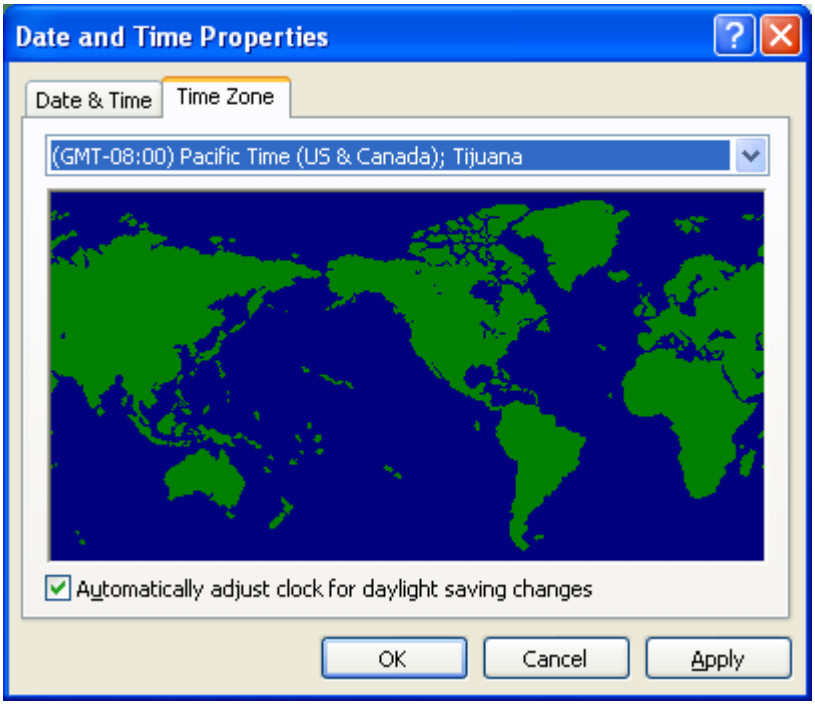
Tags / Attributes / Parameters	Phase 1	Phase 2
Control Data Path parameter	Absence of this command line parameter (**/UFO_Sync=<path>**) in the startup (.bat) file, causes Phase 1 synchronization.	The presence of this command line parameter (**/UFO_Sync=<path>**) signals Phase 2 synchronization and specifies the directory path to the shared file and, optionally, the file name.
Control Tags	Six PI tags	Five PI tags
	Phase 2 does not require both input and output tags because they are not serviced by the interface client. Phase 2 failover requires only a single Active ID, Heartbeat 1, and Heartbeat 2 tag.	
	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 DeviceStatus tags to convey the status of communications link to the data source.	
		DeviceStatus 1
		DeviceStatus 2
	[Optional] State tags	[Optional] State tags
Instrument Tag attributes	Phase 1 requires these tag attributes to communicate directly with the data source device.	
ExDesc Tag 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 tags to remove input/output designations OR Create new Phase 2 tags that do not have input/output qualifiers.
3.	Create DeviceStatus 1 and DeviceStatus 2 tags.
4.	Create new tags for Phase 2 or change the ExDesc attribute keywords for the tags 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,



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

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

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

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

PI Data Archive v3.3 and Higher

Security configuration using piconfig

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

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

For the above,

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

```
a_trust_name
```

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

```
192.168.100.11
```

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

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

Security Configuring using Trust Editor

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

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

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 PI Interface node *as it is seen by PI Data Archive*.

Chapter 13. Starting / Stopping the Interface

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

Starting Interface as a Service

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

```
Tid3.exe /start
```

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 `pipc.log` file, Windows Event Viewer, or other sources of log messages. See the section Appendix A: Error and Informational Messages for additional information.

Stopping Interface Running as a Service

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

```
Tid3.exe /stop
```

The service can be removed by:

```
Tid3.exe /remove
```

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

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 PI 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 PI 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 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 PI 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 PI Collective.

If an Interface Node runs multiple interfaces, and these interfaces send data to two or more PI 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 PI 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 data validation and data compression, but the description of these tasks is beyond the scope of this document.)

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

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

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

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

Buffering and 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 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:

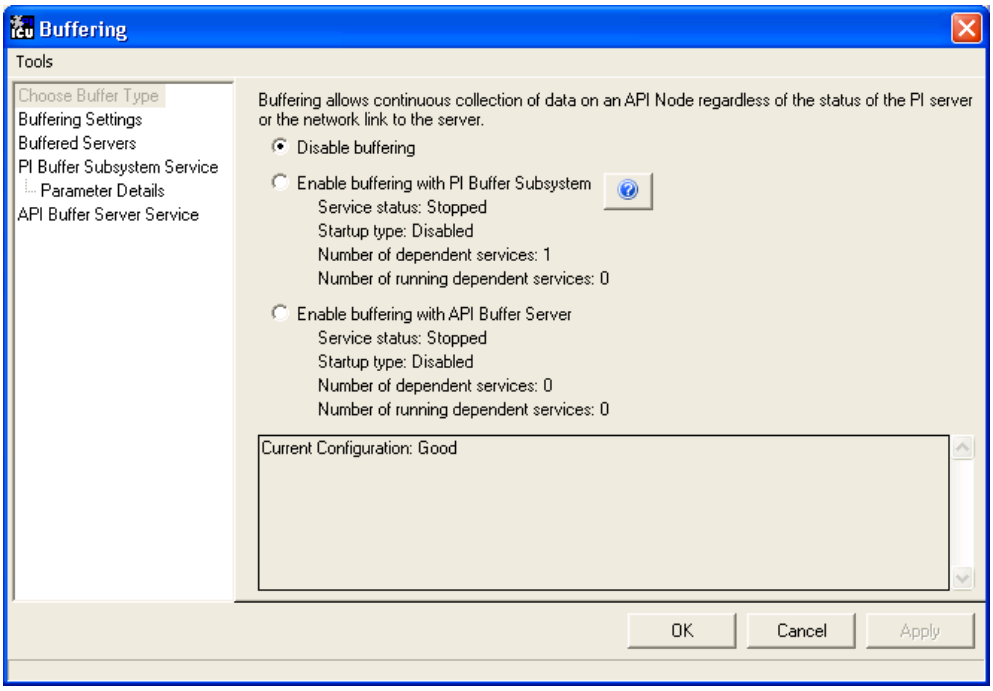
Buffering Application	Application Name field for PI Trust
PI Buffer Subsystem	PIBufss.exe
PI API Buffer Server	APIBE (if the PI API is using 4 character process names) APIBUF (if the PI API is using 8 character process names)

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.

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*.

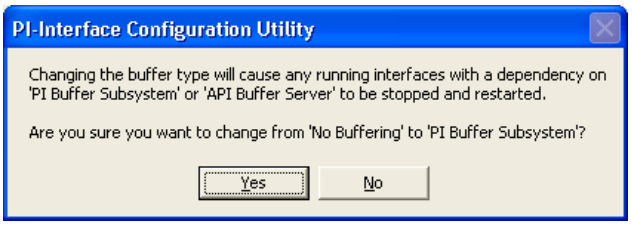
Choose Buffer Type



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*.

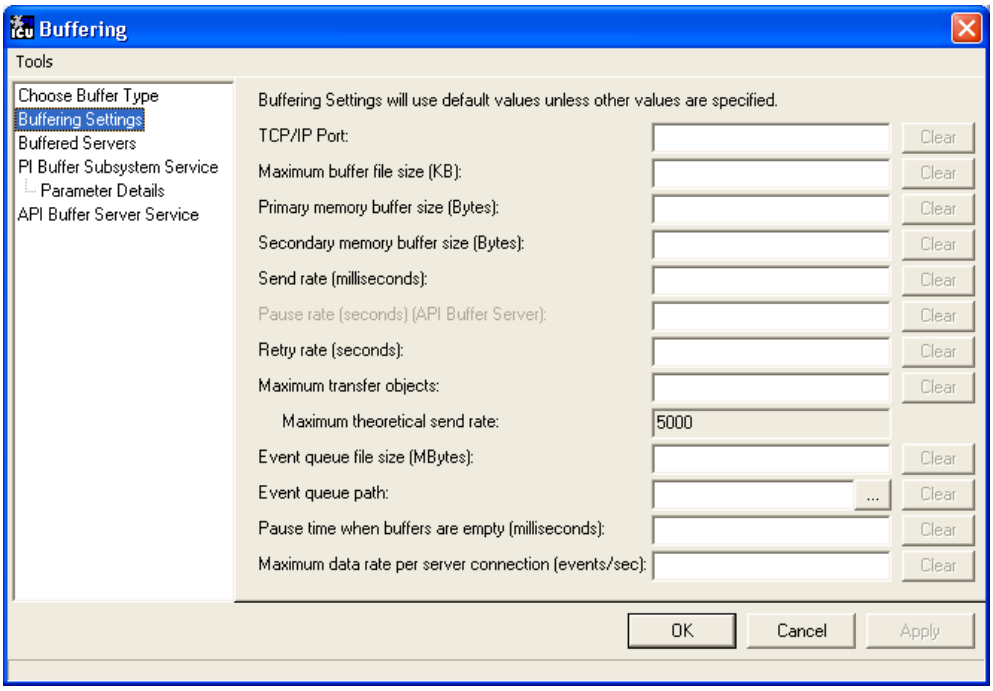


Buffering Settings

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.



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 (8 to 128 Gbytes). Please see the section entitled, "Queue File Sizing" in the *PIBufss.chm* file for details on how to appropriately size the event queue files.

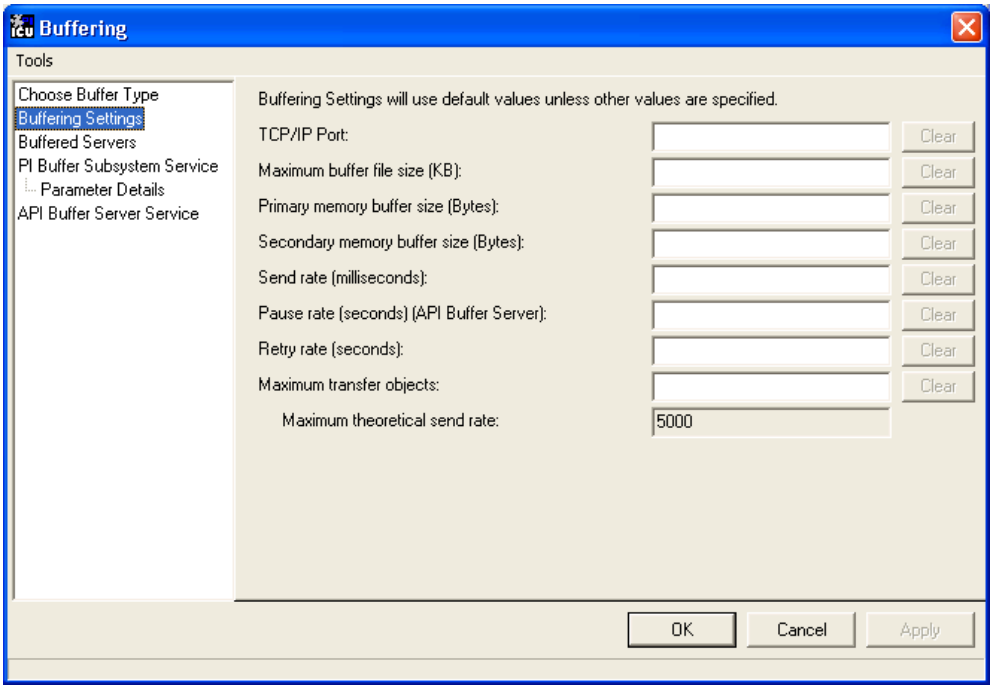
Event Queue Path

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

For optimal performance and reliability, 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.



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.

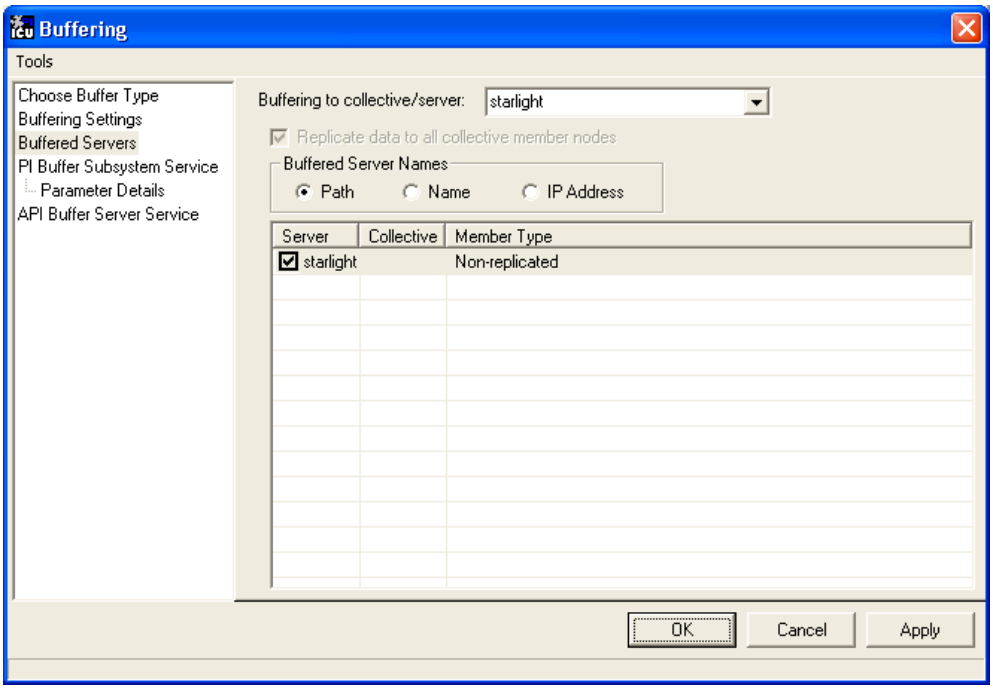
Buffered Servers

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

PIBufss

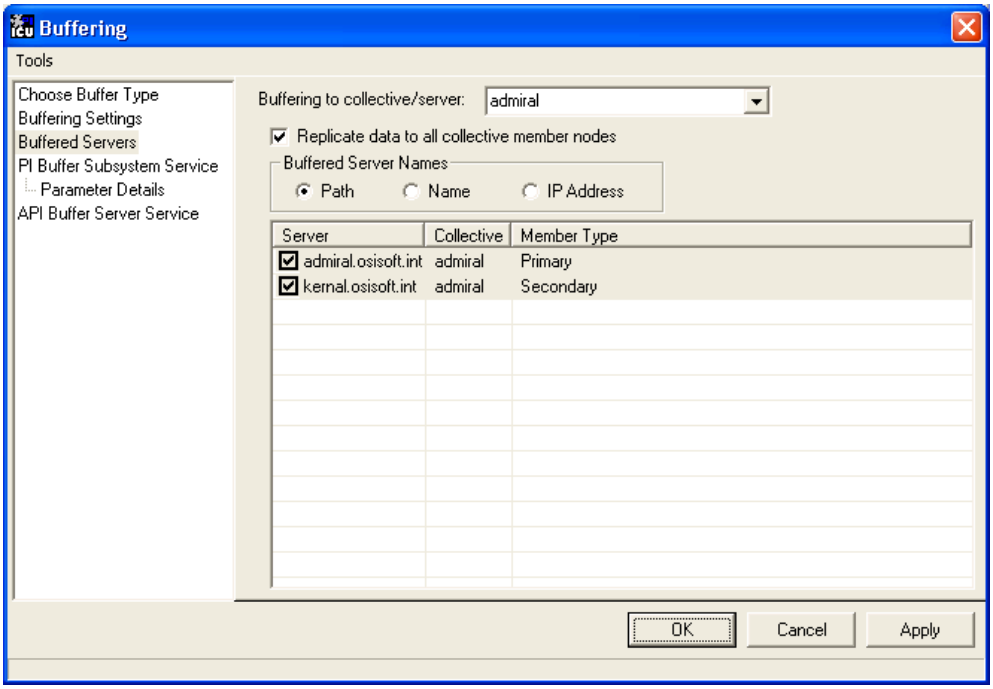
PIBufss buffers data only to a single PI Data Archive or a PI Collective. Select the PI Data Archive or the PI Collective from the *Buffering to collective/server* drop down list box.

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



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

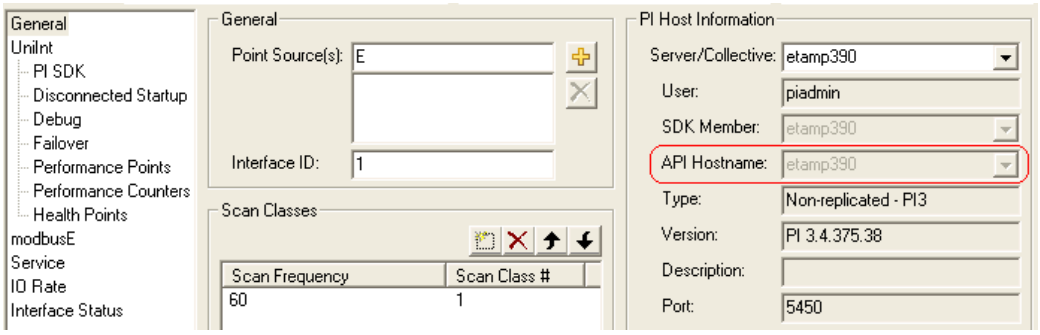
You can override this option by not checking the *Replicate data to all collective member nodes* check box. Then, uncheck (or check) the PI Data Archive collective members as desired.



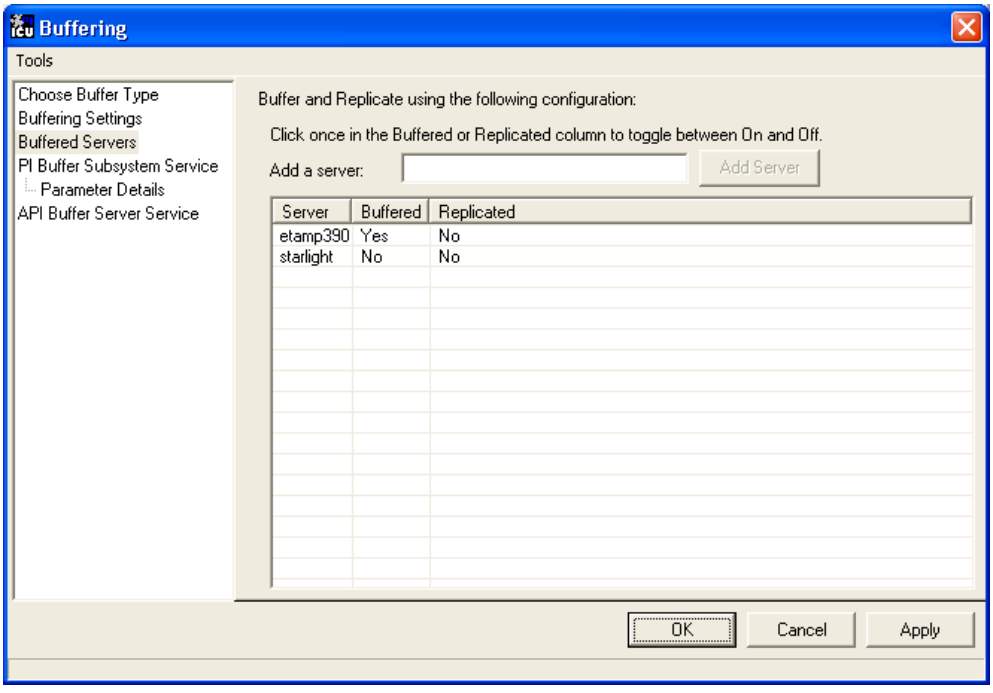
Bufserv

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 PI Collective, you should use PIBufss.)

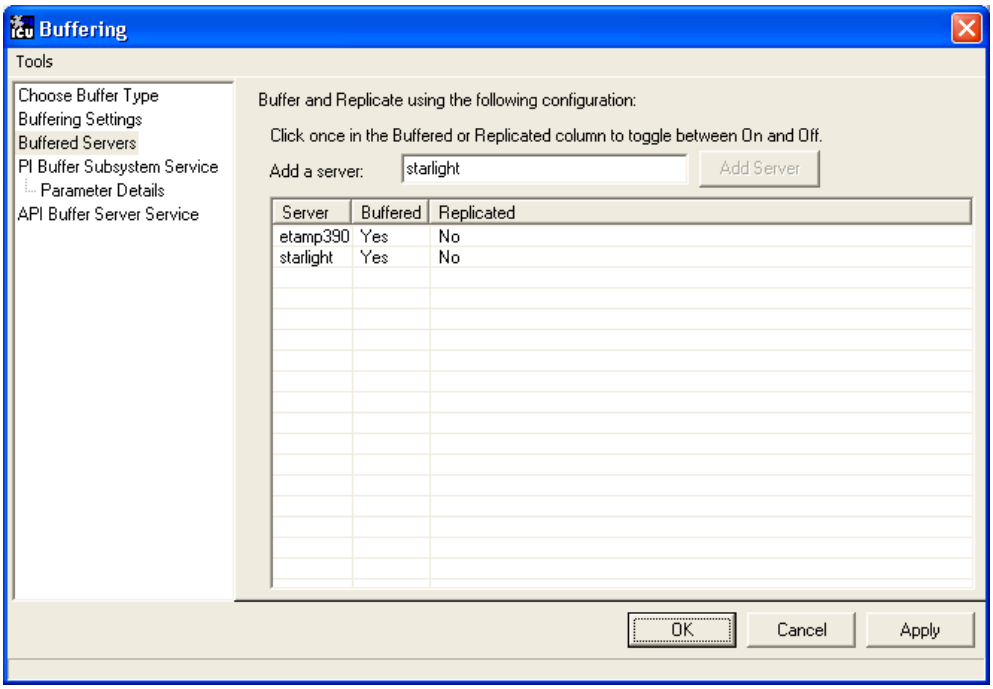
If the PI Data Archive to which you want Bufserv to buffer data is not in the Server list, enter its name in the *Add a server* box and click the *Add Server* button. This PI Data Archive name must be identical to the **API Hostname** entry:



The following screen shows that Bufserv is configured to write to a standalone PI Data Archive named `etamp390`. You use this configuration when all the interfaces on the Interface Node write data to `etamp390`.



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`.



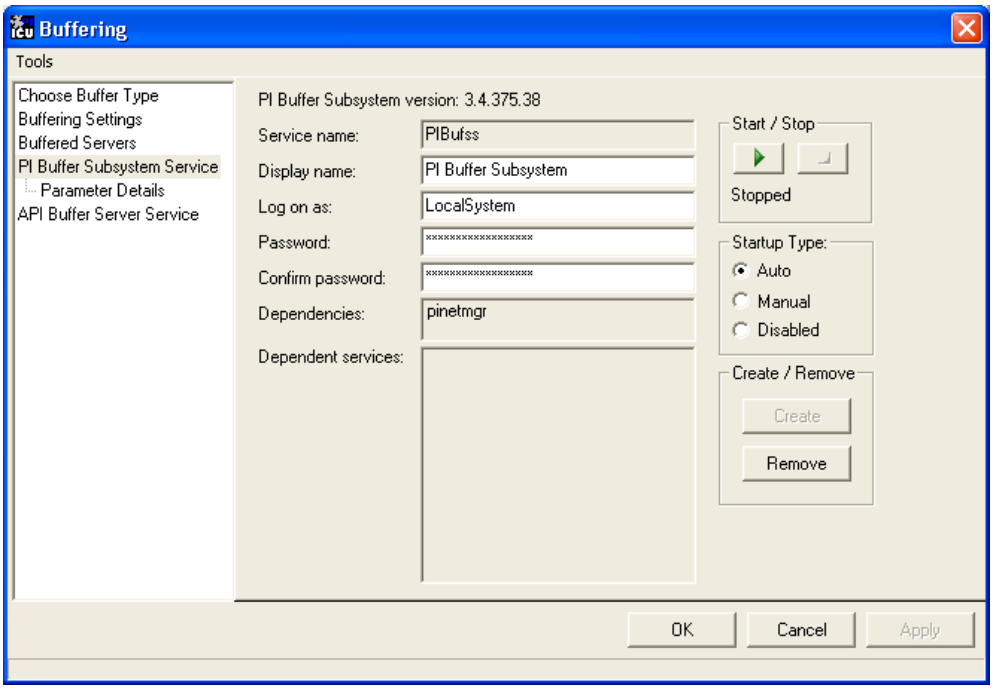
Installing Buffering as a Service

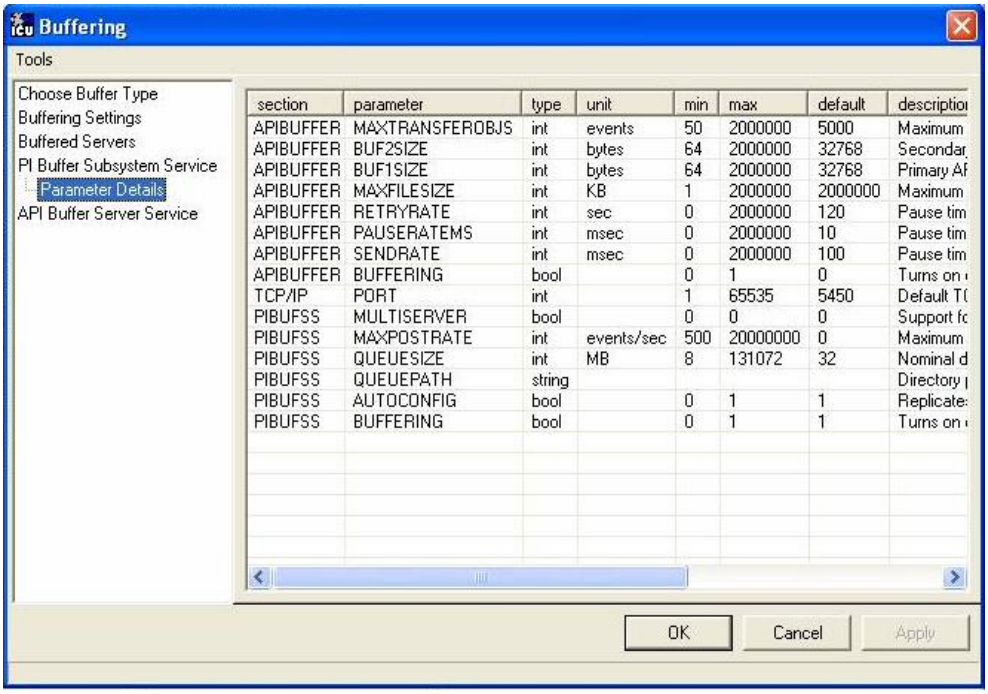
Both the PIBufss and Bufserv applications run as a Service.

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.

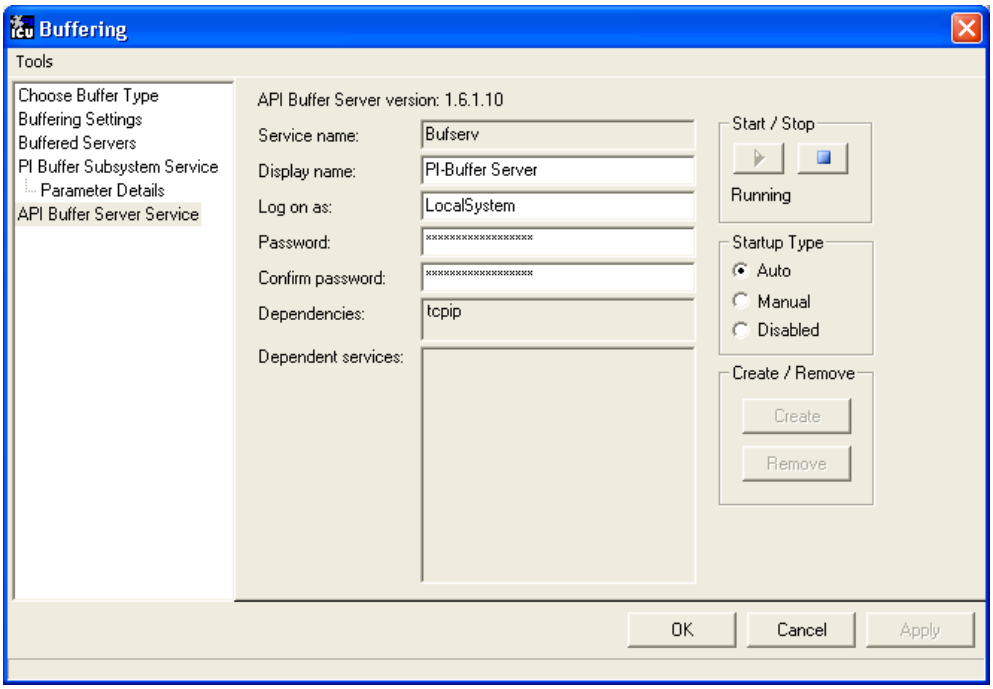




API Buffer Server Service

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.

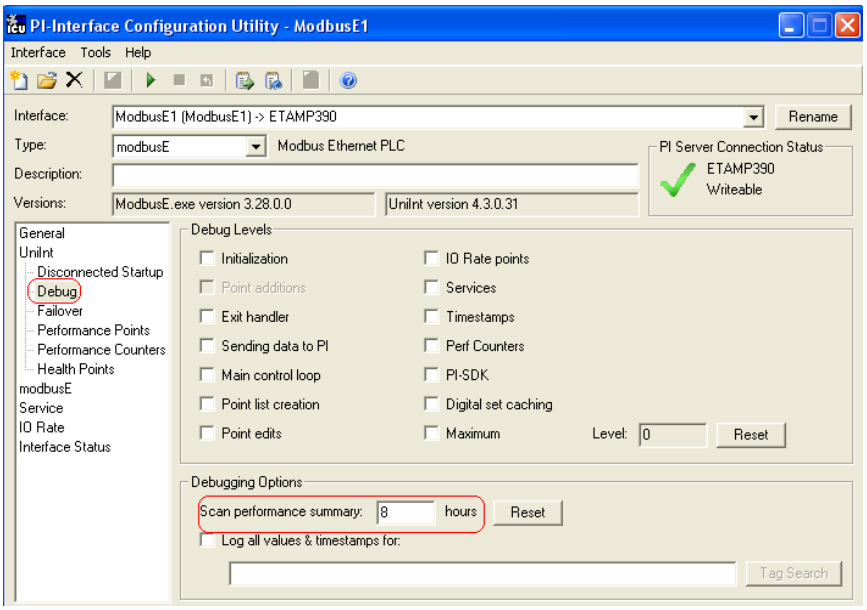

Buffering
Tools
Choose Buffer Type
Buffering Settings
Buffered Servers
PI Buffer Subsystem Service
Parameter Details
API Buffer Server Service
API Buffer Server version: 1.6.1.10
Service name:
Bufserv
Display name:
PI-Buffer Server
Log on as:
LocalSystem
Password:
Confirm password:
Dependencies:
tcpip
Dependent services:
Start / Stop
Running
Startup Type
Auto
Manual
Disabled
Create / Remove
Create
Remove
OK
Cancel
Apply

Chapter 15. Interface Diagnostics Configuration

The Interface 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**.

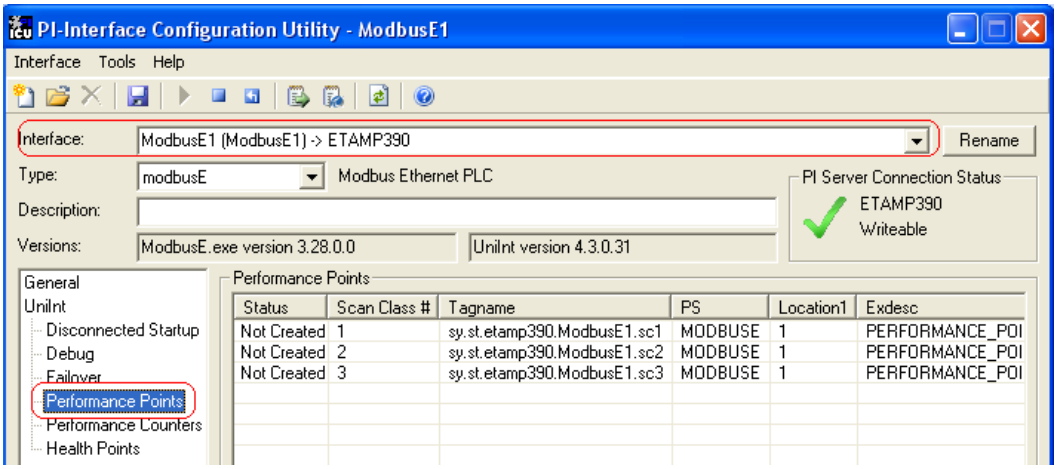
Some of the points that follow refer to a "performance summary interval". This interval is 8 hours by default. You can change this parameter via the *Scan performance summary* box in the *UniInt - Debug* parameter category pane:



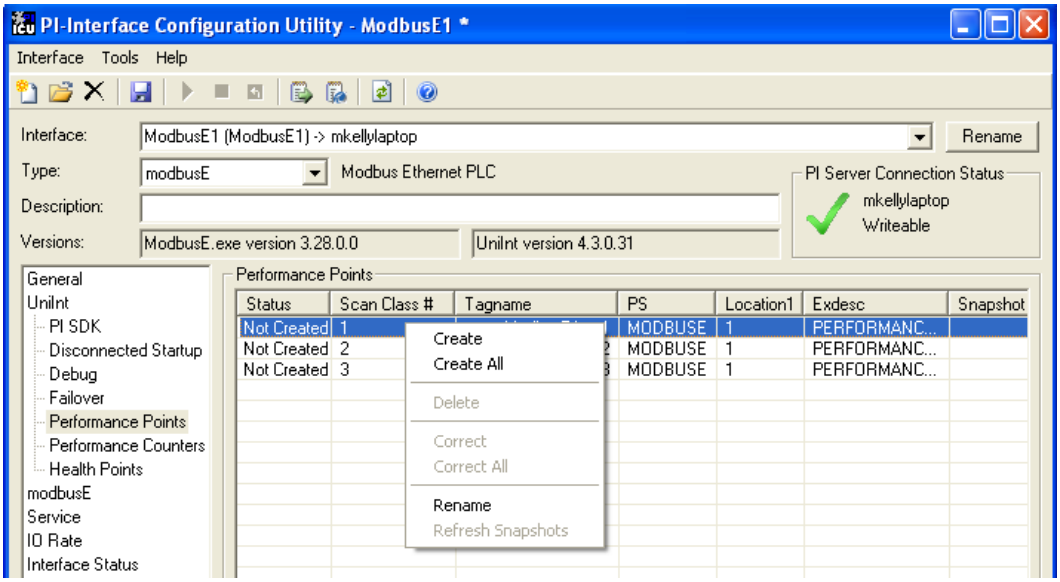
Scan Class Performance Points

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:



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



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

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

Create / Create ALL

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

Delete

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

Correct / Correct All

If the "Status" of a point is marked "Incorrect", the point configuration can be automatically corrected by ICU by right-clicking on the line belonging to the tag to be corrected, and selecting *Correct*. The Performance Points are created with the following 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	Point Source for tags for this interface, as specified on the first tab
Compressing	Off
Excmax	0
Descriptor	*Interface name* + " Scan Class # Performance Point"

Rename

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

Column descriptions

Status

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

Created - Indicates that the Performance Point does exist

Not Created - Indicates that the Performance Point does not exist

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

Scan Class

The *Scan Class* column indicates which scan class the Performance Point in the *Tagname* column belongs to. There will be one scan class in the *Scan Class* column for each scan class listed in the *Scan Classes* 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 PI. The *Snapshot* column is updated when the *Performance Points/Counters* tab is clicked, and when the interface is first loaded. You may have to scroll to the right to see the snapshots.

Performance Counters Points

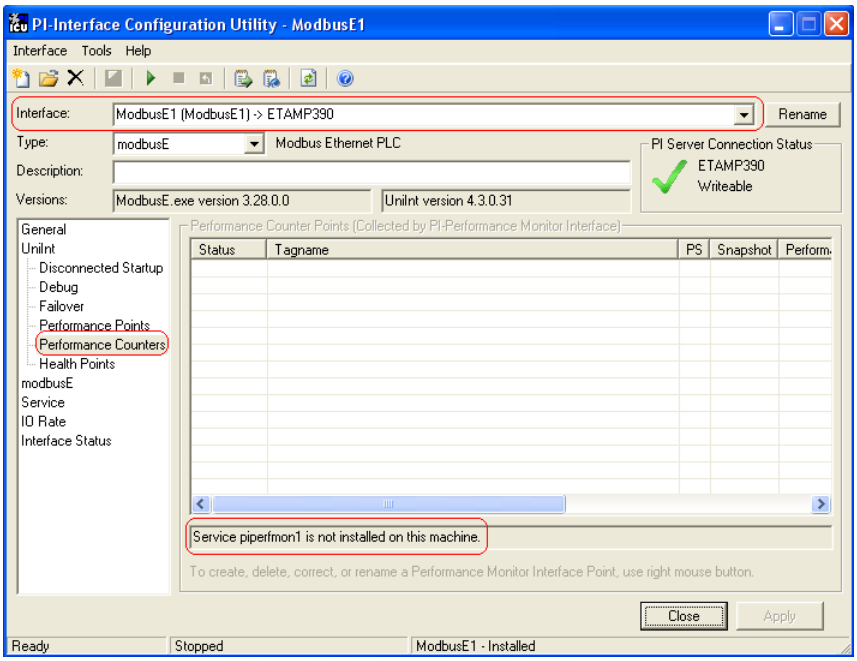
When running as a Service or interactively, this Interface exposes performance data via Windows Performance Counters. Such data include items like:

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

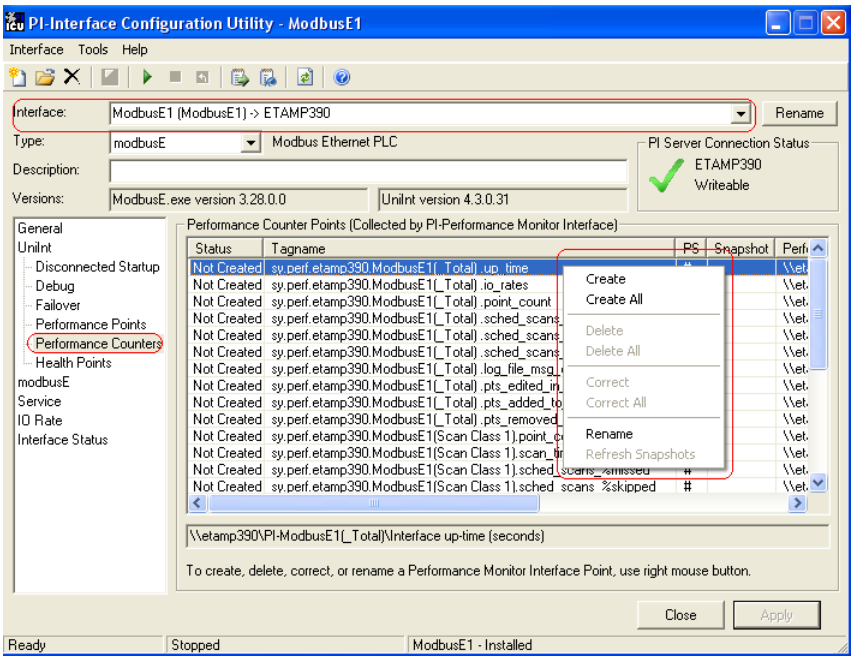
There are two types or instances of Performance Counters that can be collected and stored in PI Points. The first is (_Total) which is a total for the Performance Counter since the interface instance was started. The other is for individual Scan Classes (Scan Class x) where x is a particular scan class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



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:



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 PI. 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 tags. (As of UniInt 4.5.0.x and later this performance counters point will no longer be available.)

"Log file message count" (.log_file_msg_count)

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

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

"Points In Error" (.Points_In_Error)

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

"Points removed from the interface" (.pts_removed_from_interface)

The *.pts_removed_from_interface* Performance Counters Point indicates the number of points that have been removed from the Interface configuration. A point can be removed from the interface when one of the tag properties for the interface is updated and the point is no longer a part of the interface configuration. For example, changing the point source, location 1, or scan property can cause the tag to no longer be a part of the interface configuration.

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

The *.Points_Stale_10min* Performance Counters Point indicates the number of good points that have not received a new value in the last 10 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 PI. This counter does not include the amount of time to send the data to PI. 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)

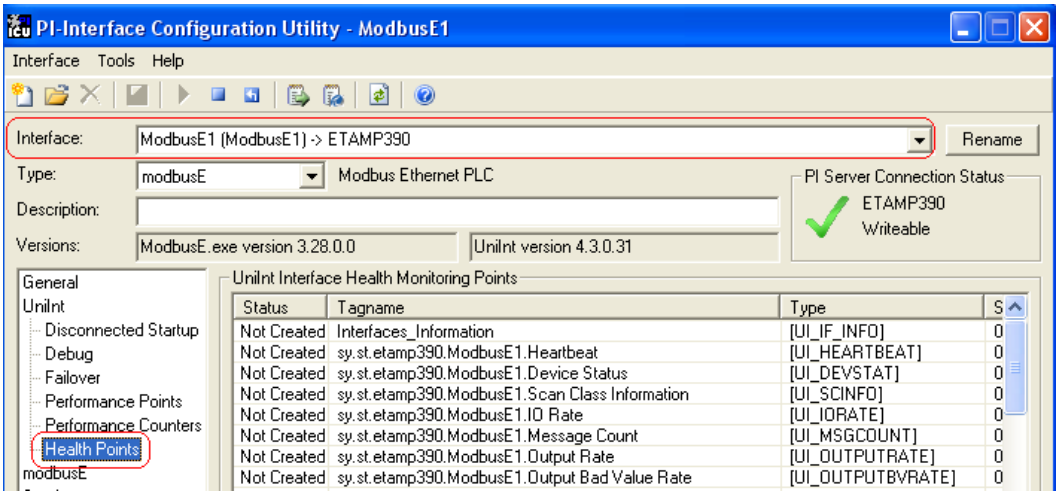
A *.scan_time* Performance Counter Point is available for each Scan Class of this Interface.

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 PI. This point is similar to the [UI_SCINSCANTIME] Health Point.

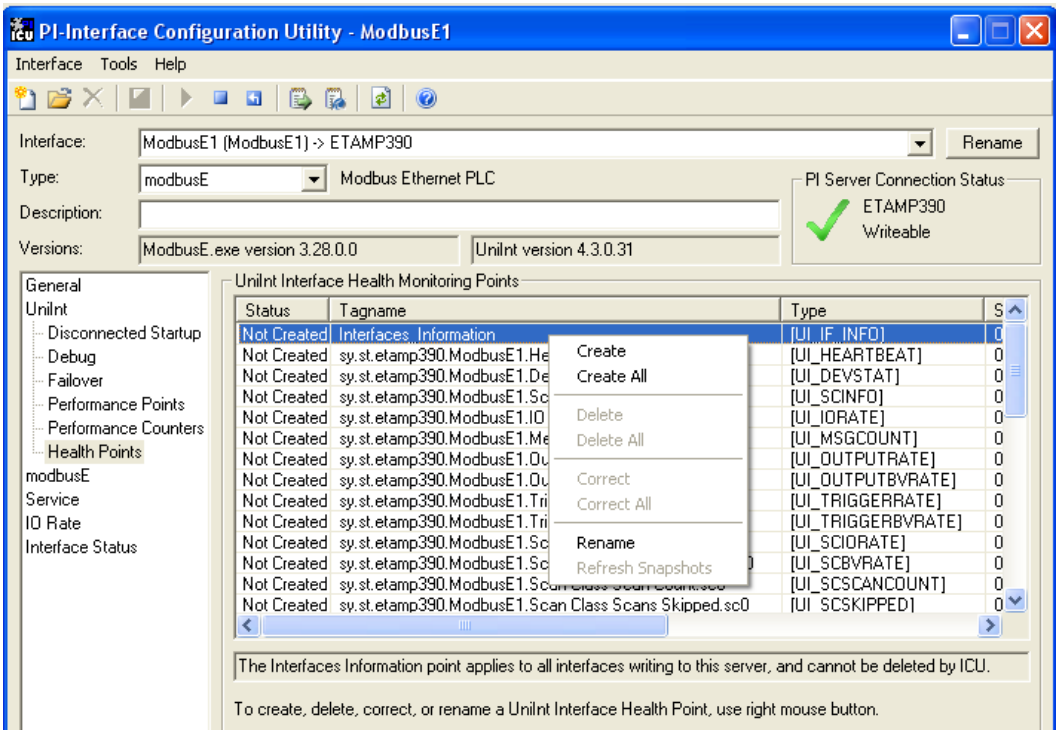
The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).scan_time`" refers to Scan Class 1, "(Scan Class 2)" refers to Scan Class 2, and so on.

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:



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



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

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

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

[UI_HEARTBEAT]

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

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

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

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

[UI_DEVSTAT]

The NovaTech D/3 Interface is built with the UniInt version that supports device status tags. This interface supports health tags. The Health tag with the point attribute Exdesc = [UI_DEVSTAT] represents the status of the source device. The following events can be written into this tag:

- "1 | Starting" - the interface is starting.
- "Good" - the interface is properly communicating and reading data from the NovaTech D/3 DBA.
- The following event represents a failure to communicate with the NovaTech D/3 DBA (the system is not running):

  "3 | 1 device(s) in error | Interface is not Connected"
- "4 | Intf Shutdown" – the interface is stopped.

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

[UI_SCINFO]

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

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

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

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

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

[UI_IORATE]

The [UI_IORATE] Health Point indicates the sum of

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

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

[UI_MSGCOUNT]

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

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

[UI_POINTCOUNT]

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

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

[UI_OUTPUTRATE]

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

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point'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 tags that have a `SourceTag`. If there are no output tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point'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 tags. If there are no event-based input tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point'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 tags. If there are no event-based input tags for this Interface, it writes the System Digital State `No Result` to this Health Point.

The Interface updates this point at the same frequency as the [UI_HEARTBEAT] point'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 tags for the scan class are no longer receiving new data.

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

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

[UI_SCBVRATE]

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

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

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

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

[UI_SCSCANCOUNT]

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

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

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

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

[UI_SCSKIPPED]

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

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

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

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

[UI_SCPOINTCOUNT]

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

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

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

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

[UI_SCINSCANTIME]

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

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

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

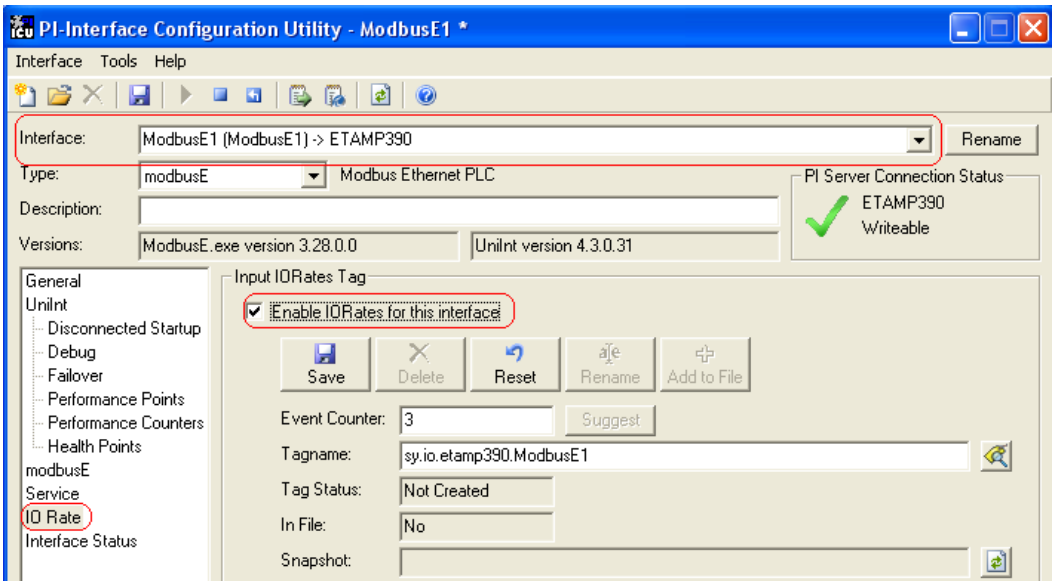
The Interface updates the value of this point at the completion of the associated scan.

I/O Rate Point

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



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

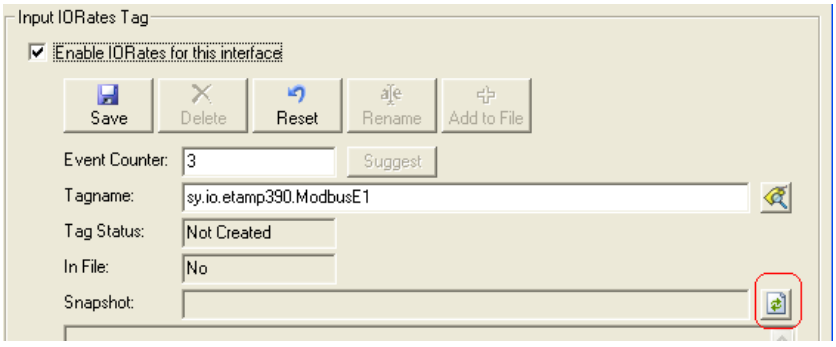
You need to restart the Interface in order for it to write a value to the newly created I/O Rate point. Restart the Interface by clicking the *Restart* button:

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

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

```
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:



Enable IORates for this Interface

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

Event Counter

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

Tagname

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

Tag Status

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

- Created – This status indicates that the tag exist in PI
- Not Created – This status indicates that the tag does not yet exist in PI
- Deleted – This status indicates that the tag 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 listed in the tag name and the event counter is in the IORates.dat file. The possible states are:

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

Snapshot

The *Snapshot* column holds the snapshot value of the I/O Rate tag, if the I/O Rate tag exists in PI. 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 tag with the tag name indicated in the *Tagname* column.

Delete

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

Rename

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

Add to File

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

Search

Allow the user to search the PI Data Archive for a previously defined I/O Rate tag.

Interface Status Point

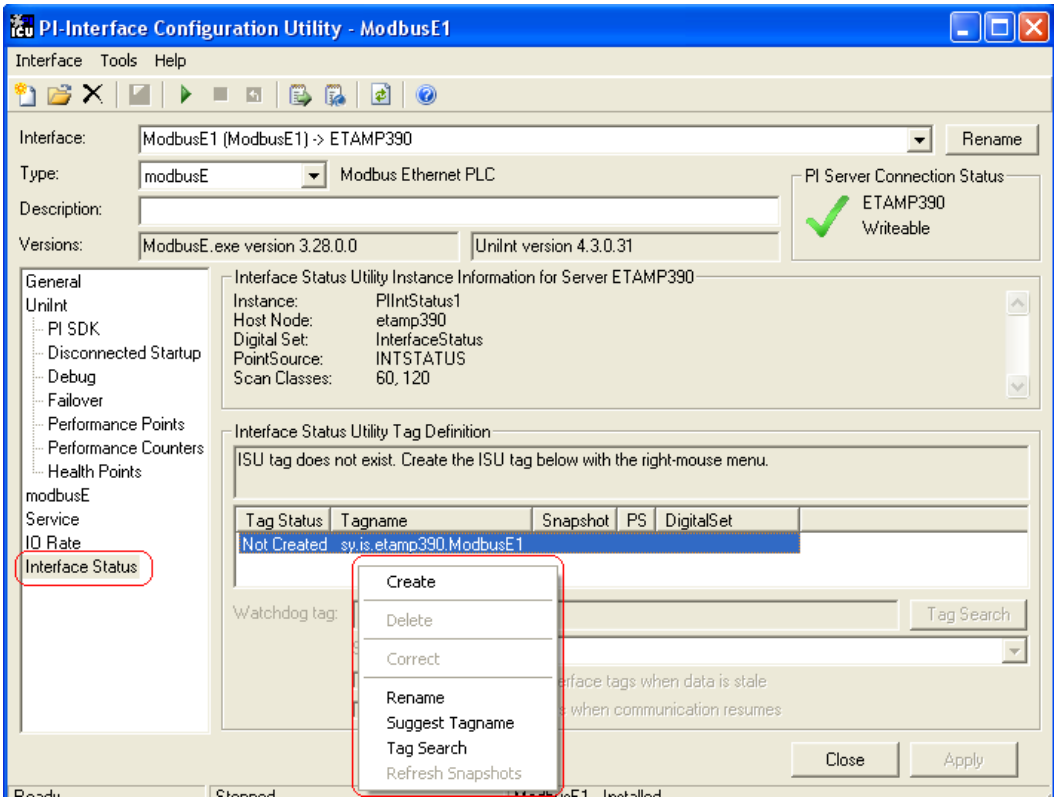
The PI Interface Status Utility (ISU) alerts you when an interface is not currently writing data to the PI Data Archive. This situation commonly occurs if

- 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 Tag. The Watchdog Tag is a tag whose value a monitored interface (such as this Interface) frequently updates. The Watchdog Tag has its `excdev`, `excmin`, and `excmax` point attributes set to 0. So, a non-changing timestamp for the Watchdog Tag indicates that the monitored interface is not writing data.

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

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



Click *Create* to create the ISU tag.

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

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

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

Note: The PI Interface Status Utility – and not this Interface – is responsible for updating the ISU tag. 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. See the *UniInt Interface User Manual* for more information.

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

- When the Interface starts many informational messages are written to the log. These include the version of the interface, the version of UniInt, the command-line parameters used, and the number of points.
- As the Interface 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

Message	`23-Jul-10 09:26:10`
	`PI_D3 2> GLF Failure, error = -69  Unit = 12, DBTYPE = 6`
	`23-Jul-10 09:26:10`
	`PI_D3 2> Bad EPN (CC0628D-07) for tag CC0628D-07.V, error = -50`

Meaning	These errors that will be seen if a bad epn is configured in PI but not found in the D/3 server.

System Errors and PI Errors

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

Error Descriptions

On Windows systems, 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_D3 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_D3 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_D3 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_D3 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 tag is not configured properly.
Resolution	Check validity of point attributes. For example, make sure Location1 attribute is valid for the interface. All failover tags must have the same PointSource and Location1 attributes. Modify point attributes as necessary and restart the interface.

Message	16-May-06 17:38:06 PI_D3 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 tag is not configured properly.
Resolution	Check validity of point attributes. For example, make sure Location1 attribute is valid for the interface. All failover tags must have the same PointSource and Location1 attributes. Modify point attributes as necessary and restart the interface.

Message	17-May-06 09:06:03 PI_D3 > 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_D3 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_D3 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_D3 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 tag 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 tags 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_D3 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 point attributes. For example, make sure Location1 attribute is valid for the interface. All failover tags must have the same PointSource and Location1 attributes. Modify point attributes as necessary and restart the interface.

Message	17-May-06 09:06:03 PI_D3 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, if the value for the Heartbeat point shows up as "***" on the D/3 Console Panel, then modify the value of the point to a valid value.

Errors (Phase 2)

Unable to open synchronization file

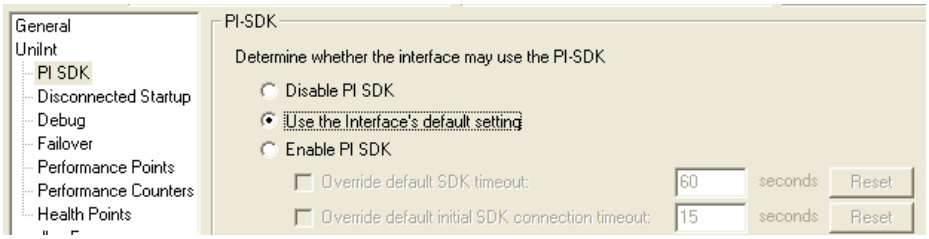
Message	27-Jun-08 17:27:17 PI Eight Track 1 1> Error 5: Unable to create file '\\georgiaking\GeorgiaKingStorage\UnIntFailover\\PIEightTrack_eight_1.dat' Verify that interface has read/write/create access on file server machine. Initializing UniInt library failed Stopping Interface
Cause	This message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



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. 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 PI 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 PI 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 compressions 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 Collective

A PI 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.

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 writes informational, debug and error message. 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 *UniInt Interface Message Logging.docx* file 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 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.

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 a PI Interface Node.

Pipc.log

The `pipc.log` file is the file to which OSIsoft applications 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 foreign device. For example, a single "point" on the foreign device can consist of a set point, a process value, an alarm limit, and a discrete value. These four pieces of information require four separate PI points.

Service

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

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

Tag (Input Tag and Output Tag)

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

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

Appendix D. 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 E. Revision History

Date	Author	Comments
02-Jul-1997	CGoodell	Added more information for compiling and linking.
08-Sep-1998	CGoodell	Elaborate on location 5, update command files
14-Sep-1998	CGoodell	Added FTP section; D/3 point configuration
15-Sep-1998	CGoodell	Added /CHANGE and /HARDWARE for compile
02-Oct-1998	CGoodell	Version 3.0; changed info about recovery instead of exit
13-Nov-1998	CGoodell	Version 3.18; if D3 AST routine called for shutdown, no digital state written to input tags; updated piftp.com; added revision significant changes
26-Mar-1999	CGoodell	Version 3.19; OSI now provides object files
08-Apr-1999	CGoodell	Added control of digital states to outputs
16-Jul-1999	HOrlett	Modified PIFTP to use consistent directories.
29-Sep-1999	CGoodell	Version 4.0 runs on Windows; use UniInt 3.x for string support
01-Oct-1999	CGoodell	Added /KS switch information.
17-Apr-2000	CGoodell	Fixed an error in which file to rename
17-Jan-2001	CGoodell	Removed /file and /temp
10-Oct-2001	CGoodell	Updated list of install files
11-Dec-2001	CGoodell	Formatting changes; updated version number; fixed headers and footers
04-Jun-2002	CGoodell	Skeleton 1.11; ICU, output strings, version 4.27
03-Jun-2003	CGoodell	Added TID3_10_1.exe and TID3_11_2.exe; increased version to 4.29.0.0.
28-Oct-2004	CGoodell	Version 4.29.0.2 Rev A: Added /itdelim
15-Mar-2006	MMoore	Updated to use Skeleton 2.0
17-Apr-2006	BPayne	Added Failover section
26-Apr-2006	MMoore	Update debug levels and command-line parameters
15-May-2006	MMoore	Added Failover Appendix; Updated to v 4.30.0.12
05-Jun-2006	MMoore	Updated to use Skeleton 2.5
19-Jun-2006	MMoore	Updated to use Skeleton 2.5.1
21-Jun-2006	BPayne	v4.30.0.12 –Updated DCS control point section with information about creating DCS EPNs and downloading.
27-Jun-2006	BPayne	Rev B.- Added updates from Interface Manual Skeleton v2.5.2.

Date	Author	Comments
06-Jul-2006	Janelle	Version 4.30.0.12 Revision C: updated headers and footers, added missing information for field length for ExDesc and InstrumentTag; removed first person references; fixed broken links; updated hardware diagram; assigned correct template to document; fixed file properties.
11-Jul-2006	BPayne	Version 4.30.0.12 Revision D: Included the name of the sample SABL program name; PI.MDL.txt, that is provided for assisting in configuring D/3 failover control EPNs.
18-Jul-2006	BPayne	Version 4.30.0.12 Revision E: Removed all references to UFO_Interval as it is not used with scan based only interfaces such as the Novatech D/3.
12-Nov-2007	MMoore	Version 4.3.1.15: Updated to Skeleton 2.5.6
12-Dec-2007	Janelle	Version 4.3.1.15, Revision A: updated some formatting, added Disconnected Startup and Set Device Status information in the Supported Features table.
13-Dec-2007	MMoore	Version 4.3.1.15, Revision B: removed VMS information, updated Device Status and Disconnected Startup
14-Dec-2007	MMoore	Version 4.3.1.15, Revision C: removed PI2/VMS, Added 12.2-3
14-Dec-2007	Janelle	Version 4.3.1.15, Revision D: updated table formatting
17-Sep-2008	BPayne	Version 4.30.2.16, Rev A
13-Oct-2008	MMoore	Version 4.30.2.17, Rev A: Updated to use Skeleton Version 3.0.4
11-Dec-2008	BPayne	Version 4.30.2.17, Rev B: Updated to use Skeleton Version 3.0.7
29-Jan-2009	Janelle	Version 4.30.2.17, Revision C: updated headers, updated hyperlinks. Updated copyright date, updated supported Operating Systems, updated table formatting
16-Feb-2009	Janelle	Version 4.30.2.17, Revision D: updated UniInt Failover Phase 2 Control Tags to match current ICU functionality. Fixed broken reference.
01-Nov-2010	BPayne	Version 4.31.4.31, Revision A: Updates to use Skeleton Version 3.0.28.
15-Apr-2011	SBranscomb	Version 4.31.4.31, Revision B; Updated to Skeleton Version 3.0.32.
03-Jun-2011	BPayne	Version 4.31.5.x, Revision A: Removed references to Windows OS and set startup arguments /ch and /hw to “recommended”
12-Jun-2014	MKelly	Version 4.31.5.x, Revision B: Updated to Skeleton Version 3.0.38
16-Jun-2014	ASudhakaran	Version 4.31.7.x, Revision A: Updated - Supported OS section
10-Sep-2014	JMuraski	Version 4.31.7.x, Revision B: Updated per Release Team requests