PI Interface for Metso maxDNA

Version 1.5.1.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 Metso maxDNA

Published: 01/2014

Table of Contents

Chapter 1. Introduction

The PI Interface for Metso maxDNA (formally known as the Max Controls Max1000 Plus+ Interface) collects data from the maxDNA members of a maxDNA system. This interface will be referred to as the maxDNA interface or, simply, the interface for the remainder of this document.

Note: The interface requires that the maxDNA software be present on the same PC as the interface and the PC have network access to the SBP.

Migration interfaces are available to connect the PI System to all generations of MAX systems.

Note: Contact Metso Automation to run a system analysis to determine available throughput on older systems.

For proper interface operation, configure points (tags) on the home node (the words "point" and "tag" are used interchangeably in this manual). Tags are used to update and receive data from maxDNA members. A single interface can collect data from one or more maxDNA members at a time. Data is received at a given frequency. All values that are written to the snapshot or archive use the system time from the PI Server node.

At startup, the interface scans the PI Point Database for all associated points and builds its own point list. During runtime, the interface continues to check the PI Point Database for point updates and modifies its point list accordingly. If the Scan field of any point on the point list is set to off, the point is removed from the point list. The point is added once again after the Scan field is turned back on. If a fixed scan rate cannot be found for a given point, the point will be removed from or will not be added to the point list.

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 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.

Reference Manuals

OSIsoft

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

Vendor

- *maxDNA Installation Instructions*

Supported Operating Systems

Platforms		32-bit application	64-bit application
Windows XP	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 2003 Server	32-bit OS	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 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	Yes	No
	64-bit OS	Yes (Emulation Mode)	No
Windows 2012	64-bit OS	Yes (Emulation Mode)	No

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

The version 4.x series maxDNA software requires Windows XP SP2 to run.

Please contact OSIsoft Technical Support for more information.

Supported Features

Feature	Support
Interface Part Number	PI-IN-MCS-PLUS-NTI
Auto Creates PI Points	No
Point Builder Utility	No
ICU Control	Yes
PI Point Types	float16, float32, float64, int16, int32, digital
Sub-second Timestamps	No
Sub-second Scan Classes	No
Automatically Incorporates PI Point Attribute Changes	Yes
Exception Reporting	Yes
Outputs from PI	Yes
Inputs to PI:	Unsolicited
Supports Questionable Bit	No
Supports Multi-character PointSource	Yes
Maximum Point Count	Unlimited
* Uses PI SDK	No
PINet String Support	N/A
* Source of Timestamps	PI Server
History Recovery	No
* UniInt-based * Disconnected Startup * SetDeviceStatus	Yes Yes Yes
* Failover	UniInt Failover (Phase 2- Warm, Cold); Software Backplane Failover
* Vendor Software Required on Interface Node / PINet Node	Yes
Vendor Software Required on Foreign Device	Yes
Vendor Hardware Required	Yes
Additional PI Software Included with interface	No
Device Point Types	int16, int32, float16, float32, float64, digital
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

Timestamps are generated on PI Server.

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 maxDNA 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 Server. 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 disconnected startup.

SetDeviceStatus

Failover

- Software Backplane Failover
    - The Software Backplane handles failover from one DPU to another. In most cases, the interface needs to re-subscribe its points after a failover. Virtual DPU as well as physical DPU failover have been tested.

Note: When recovering a failed virtual DPU in primary/backup mode, make sure the backup DPU is started completely before starting the primary DPU. This is to make sure the primary DPU is able to load its point list correctly. Otherwise, re-subscribe attempts will fail.

- UniInt Failover Support

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

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

Vendor Software Required

Software Backplane is the generic name given to the software that runs between the interface and the DPUs. It consists of several parts. The relevant ones for this interface are maxAPPS and maxSTATION. maxSTATION is the full version of the software distributed by Metso Automation (formerly MAX Controls). It runs on its own machine. maxAPPS is the software that is needed on the interface node.

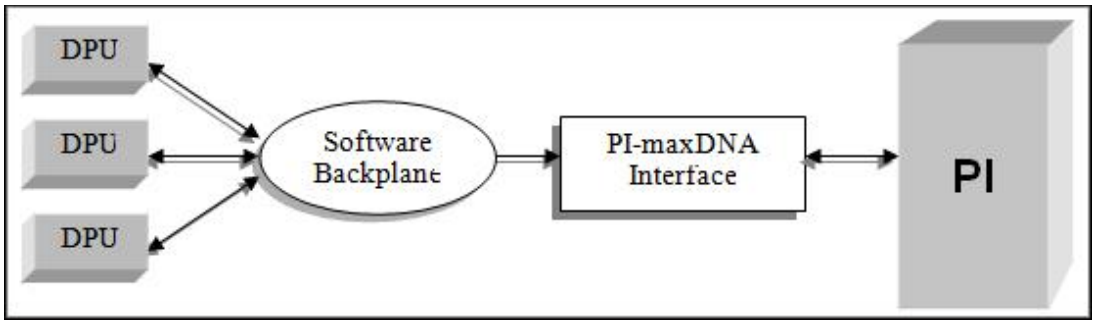
Vendor Hardware Required

The Software Backplane connects to the DPUs that are used to collect plant data. The DPUs can be physical DPUs or virtual DPUs (software emulation of hardware).

Device Point Types

The Software Backplane supports int16, int32, float16, float32, and float64 point types for data, and digital types for quality reporting.

Diagram of Hardware Connection



Chapter 2. Principles of Operation

The maxDNA SBP uses "subscriptions" to mark data points for frequent update. The interval between updates is configurable on a point-by-point basis. The maxDNA system only sends data to a client when an exception has occurred. An exception occurs if the value changes by more than the given dead band, or the exception maximum time has expired and the dead band (exception deviation) has not been exceeded.

When the Interface first starts up, it establishes communication with PI Server. A connection to the local maxDNA server is then established. The connection is uniquely identified by the client with a "user name" parameter, and uniquely identified to the server by an identification number. The interface to the maxDNA system can subscribe, unsubscribe, and read from specific data points. It also has the ability to write to a designated WatchDog tag on the maxDNA system, which can be optionally configured by the user. All data handled by the maxDNA SBP is done with variant data types. This means that when data is assigned to a tag, the data is automatically handled using the most compatible type. The maxDNA SBP has comparable data types to handle all types supported by the PI System.

Exception reporting is done on the maxDNA system using the PI exception parameters which are passed to it during subscription. As each point belonging to the interface is identified, the interface subscribes the maxDNA point (specified in the InstrumentTag and/or the ExDesc) on the SBP. The exception minimum (ExcMin), exception maximum (**`ExcMax`**), and exception deviation (ExcDev) (in engineering units) are also sent to the SBP. The maxDNA system will check to see if the point subscribed is valid, and if it is valid, can it be accessed. An error code is returned if the point is invalid or cannot be accessed and the interface will print a message that it could not be subscribed.

Note: Some points may not be accessible at interface startup, but once they do become accessible, data will automatically start being collected for them. Prior to data collection, the digital state CONFIGURE is written to all points. This allows the user to easily determine which points have not begun collecting data. If connection is lost to the maxDNA system, I/O Timeout is written to all the input tags. BAD INPUT is written to points when any other error is returned from the maxDNA system.
UniInt Failover Note: UniInt Failover Control Tags behave slightly different from the description above. The interface does not write CONFIGURE status to Failover Control Tags prior to data collection.

The maxDNA system uses the PI exception specifications that are passed for a point to determine when to check for an exception and determine when an exception has occurred. An exception occurs when the maxDNA tag exceeds the ExcDev for the point or the **ExcMax** time has been exceeded with no exception occurring. The ExcMin time controls the frequency at which exceptions are checked. The minimum exception minimum time passed to the maxDNA system is 1 second. If a tag has its ExcMin parameter set to 0, then a value of 1 will be passed to the maxDNA system. The maximum exception maximum time passed to the maxDNA system is 30 seconds if the **/excmax** parameter is not used. The **/excmax** parameter can be used in the interface startup file to set the maximum exception maximum parameter for all tags to a value greater than 30 seconds. If **/excmax** is not set in the interface startup file and a tag has its **ExcMax** parameter set greater than 30 seconds, then a value of 30 will be passed to the maxDNA system. If **/excmax** is set in the interface startup file and a tag has its **ExcMax** parameter set greater than **/excmax**, then the **/excmax** value will be passed to the maxDNA system. Consult your Metso Automation representative for recommendations for exception maximum settings. Typical systems will be able to handle **ExcMax** times of 30 seconds for all tags. However older systems may not.

Since exception reporting is done on the maxDNA side no further exception reporting is done within the interface. Value and quality data are sent to PI Server when they are received by the interface. Although the scan frequency is not used, the Location4 parameter for all input points must still be set to one.

UniInt Failover

This interface supports UniInt failover. Refer to the UniInt Failover Configuration chapter 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. Verify that the maxAPPS software is installed and is working correctly.
2. Confirm that you can use PI SMT to configure the PI Server. You need not run PI SMT on the same computer on which you run this interface.
3. If you are running the interface on an interface node, edit the PI Server's Trust Table to allow the interface to write data.
4. 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.
5. 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.
6. If you are running the interface on an interface node, check the computer's time zone properties. An improper time zone configuration can cause the PI Server to reject the data that this interface writes.
7. Run the ICU and configure a new instance of this interface. Essential startup parameters for this interface are:

   Point Source (**`/PS=`***`x`*)
   Interface ID (**`/ID=`***`#`*)
   PI Server (**`/Host=`***`host:port`*)
   Scan Class (**`/F=`***`##:##:##,offset`*)
8. If you will use digital points, define the appropriate digital state sets.
9. Define digital states if quality tags are being used.
10. 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 specifies whether the tag is an input (0), output (1), or watchdog (2) tag.
Location3 specifies whether the tag is a value (0) or quality (1).
Location4 specifies the scan class.
Location5 is not used.
ExDesc is not used unless the member portion of the point address is omitted in the InstrumentTag.
InstrumentTag specifies the point address on the Software Backplane.

11. Start the interface interactively and confirm its successful connection to the PI Server without buffering.
12. Confirm that the interface collects data successfully.
13. 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.
14. 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 Server Node or by stopping the PI Server.
15. Configure the interface to run as a Service. Confirm that the interface runs properly as a Service.
16. 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 Server 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 disconnected 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 on Windows

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

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

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

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

Naming Conventions and Requirements

In the installation procedure below, it is assumed that the name of the interface executable is `PIMax.exe` and that the startup command file is called `PIMax.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, `PIMax1.exe` and `PIMax.bat` would typically be used for instance 1, `PIMax2.exe` and `PIMax2.bat` for instance 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\ Max1000pp\

PIHOME is defined in the pipc.ini file.

Interface Installation Procedure

The maxDNA 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.

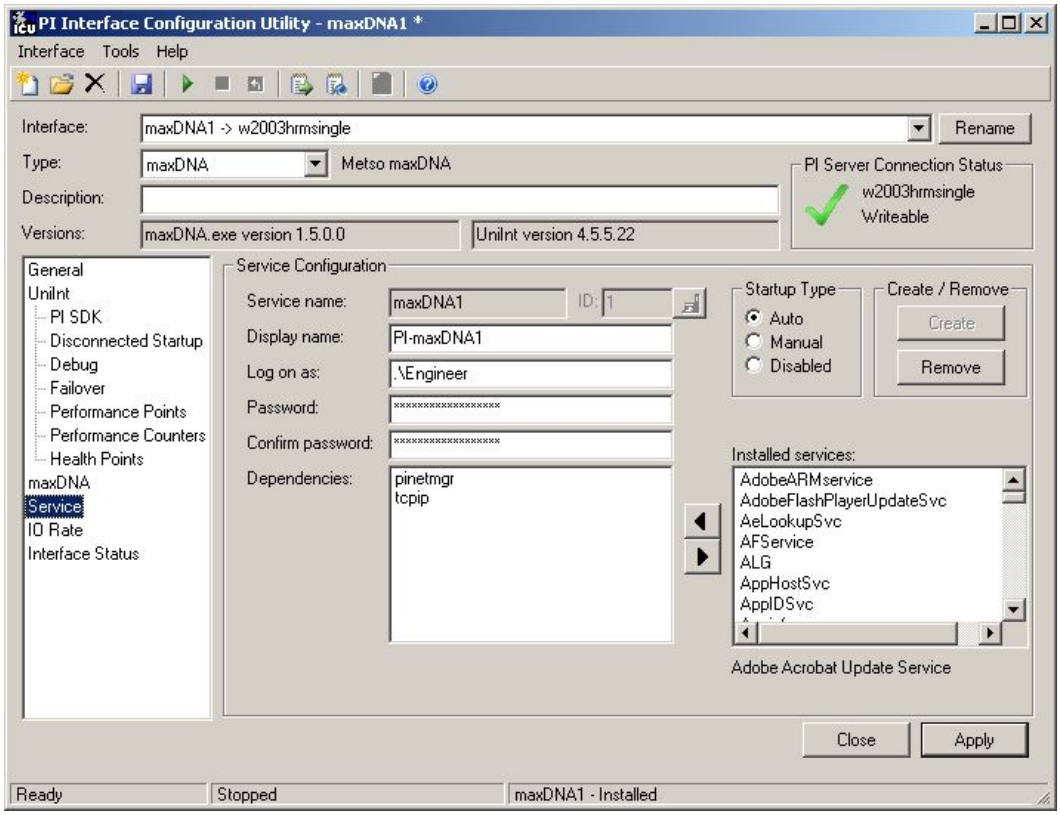
maxDNA_#.#.#.#_.exe

Installing Interface as a Windows Service

The maxDNA interface service can be created, preferably, with the PI Interface Configuration Utility, or can be created manually. The service should be run under Engineer user account.

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 name 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, select 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*.

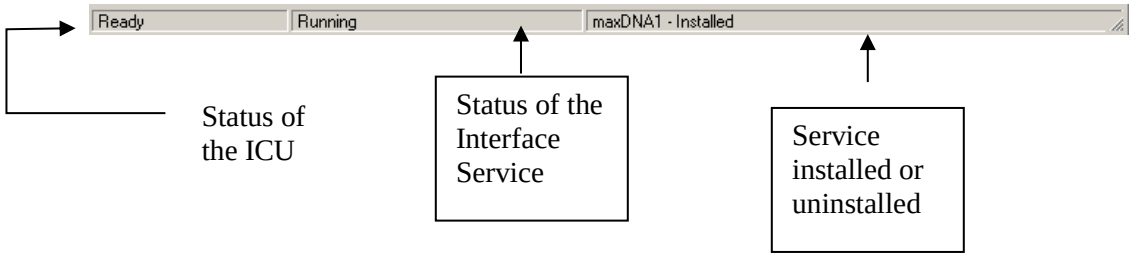
Remove

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:

```
PIMax.exe /help
```

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

Note: In the following Windows Service Installtation Commands you may use either a slash (/) or dash (-) as the delimiter.

Windows Service Installation Commands on an Interface Node or a PI Server Node with Bufserv implemented	
Manual service	`PIMax`.exe /install /depend "tcpip bufserv"

Automatic service	`PIMax`.exe /install /auto /depend "tcpip bufserv"
*Automatic service with service ID	PIMax.exe /serviceid *X* /install /auto /depend "tcpip bufserv"
Windows Service Installation Commands on an Interface Node or a PI Server Node without Bufserv implemented	
Manual service	PIMax.exe /install /depend tcpip
Automatic service	PIMax.exe /install /auto /depend tcpip
*Automatic service with service ID	PIMax.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 Server documentation.

Digital State Sets

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

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

Creation of Quality Digital State Set

You must create a digital state set for use with quality tags. A suggested name is maxDNA_QUALITIES. The digital state set must contain the states shown exactly in the order as they appear below.

```
OTHER, GOOD, NOT KNOWN, DOUBTFUL, SUBSTITUTED, BAD, BAD REF, NO VALUE
```

The value of OTHER will be given if the returned quality is not one of the other qualities shown above.

An example digital set file, `PI_maxDNA_Qualities.csv`, is provided with the interface installation kit.

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 Server 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. PI Point Configuration

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

maxDNA Tag Address Format

The PI-maxDNA Interface uses a "reference-member" (RefMem) identifier to reference a specific point in maxDNA. The reference represents the maxDNA tag group (or service) in which the tag resides. Member references the actual tag within the specified reference.

General PI Tag Configuration Information

One PI point (PI tag) must be configured for each maxDNA member you want to read from or write to. The points can be configured on a PI 2 or PI 3 home node. Each tag from the maxDNA system may also have quality along with a value. You can choose to store the quality that comes with each value in a separate PI tag.

The following describes the PI point attributes that have specific meaning for use with the PI-maxDNA Interface. Other fields may also need to be specified for proper configuration of the PI point. Some of these fields include typical value, engineering units, resolution code (PI 2 only), filter code, etc. You may also want to create I/O Rate Tags for each interface.

The attribute names used below are consistent with the names in the Data Archive Manual for PI 3.

Point Attributes

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

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

Note: See the *UniInt Interface User Manual* and PI Server documentation for information on other attributes that are significant to PI point data collection and archiving.

Tag

The Tag attribute (or tag name) 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 Server.
- The first character must be alphanumeric, the underscore (_), or the percent sign (%).
- Control characters such as linefeeds or tabs are illegal.
- The following characters also are illegal: * ' ? ; { } [ ] | \ ` ' "

Length

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

PI API	PI Server	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 Server 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 chapter.

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 and digital point types are supported. For more information on the individual PointTypes, see PI Server manuals.

Location1

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

Location2

The Location2 attribute is used to specify whether this tag is an input, output or watchdog tag. Possible values are:

0 = Input Tag

1 = Output Tag

2 = Watchdog Tag.

Location3

This attribute is used to indicate whether the PI tag will hold a value or a quality.

0 = Value tag

1 = Quality tag

Location4

Input and Output Tags

Location4 should be set to 1.

> **Note**: This interface does not support the standard trigger-based scanning that UniInt supports since all data comes from the maxDNA system on an exception basis.

Watchdog Tags

Location4 determines the frequency at which a watchdog tag will send data to the SBP. The possible scanning frequencies for a given interface are specified by the user on the command line in the `PIMAX#.bat` file (see section Startup Command File). For example, the command line is as follows:

```
/f=00:00:05 /f=00:00:15 /f=00:01:00,00:00:10
```

Then, the point can be configured to send a "heartbeat" to the SBP every 5 seconds, every 15 seconds, or every minute. For the 5-second and 15-second periods, heartbeats will begin on the hour or at a multiple of 5 or 15 seconds after the hour. For the 1-minute period, heartbeats will begin 10 seconds after the hour or at a multiple of 1 minute and 10 seconds after the hour. If Location4 is 1 for the above command line, then the watchdog tag will update every 5 seconds. If Location4 is 2, then the tag will update every 15 seconds, and so on.

Location5

Location5 is not used by this interface.

InstrumentTag

This attribute is used to specify the RefMem address for maxDNA. RefMem stands for Reference-Member, and is used to address a specific tag within maxDNA. Each entry in the RefMem must be separated with a period, with no spaces between the period and text. For example, the InstrumentTag attribute for a PI tag would contain:

```
[domain]service.member.ext.ext
```

Typically only the service and member need to be specified. An example of this type of address is: `FIC101.PV` where `FIC101` is the service and PV is the member of interest.

Length

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

PI API	PI Server	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 Server 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.

ExDesc

The ExDesc (Extended Descriptor) is used to specify the member portion of the maxDNA point address if not given in the InstrumentTag.

The member name is placed at the end of the ExDesc attribute in the following format:

```
RM=MemberName
```

This string, if required, must appear the end of the ExDesc attribute. The RM= must be given with capital letters; however the actual member name should match that given in maxDNA.

Length

Depending on the version of the PI API and the PI Server, 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 Server versions.

PI API	PI Server	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 Server 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.

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.

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 Server 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 Server 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 Server, allowing continuous data collection when the PI Server is down for maintenance, upgrades, backups, and unexpected failures. That is, when the PI Server 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 Server Collective. Refer to the Bufserv or PIBufss manuals for additional information.

Output Points

Output points control the flow of data from the PI Server to any destination that is external to the PI Server, 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 below. The only difference is that the trigger tag is specified with the SourceTag attribute instead of with the "event" or "trig" keywords. For output points, event conditions are specified in the extended descriptor as follows:

event_condition

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."

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.

Quality Points

Quality tags are declared by setting the tag's Location3 field to a 1. An input tag can then specify this quality tag in its SourceTag field. When data is sent from the SBP, its quality is then written to this quality tag.

Quality tags must be of Digital data type. Also, the DigitalSet field of a quality tag must match the digital set created for qualities as described above. Failure to meet these requirements will cause the quality tag to report erroneous data.

Watchdog Points

A watchdog tag is used as a software "heartbeat." It creates a timer in the local status server set for a 60-second timeout.

For the tag to function correctly it needs two things. The first is a service name to use as the destination. This is specified in the tag's InstrumentTag field. On startup, the interface will create a service in the local status server using the name given in this field. The second thing the tag needs is a heartbeat interval. Location4 corresponds with the desired heartbeat interval. See the description of Location4 above for a detailed explanation of specifying heartbeat intervals.

While operational, the tag automatically sets the timer to 60 seconds at the interval given in Location4. Therefore, the interval referenced by Location4 should be considerably less than 60 seconds. In the event that the interface fails to reset the timer before the 60-second time limit, an alarm will be raised in the newly created service.

The actual value stored in the PI tag when it is fully operational is the digital state Good. Should the target SBP item become unreachable, the digital state I/O Timeout is written to the watchdog tag.

While the watchdog tag is operational, one can view the current state of the timer by subscribing to the SBP item ServiceName.time left. However, you must ensure that this subscription does not occur before the service is created. Failure to meet this requirement would cause the subscription attempt to fail.

Chapter 8. Startup Command File

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

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.

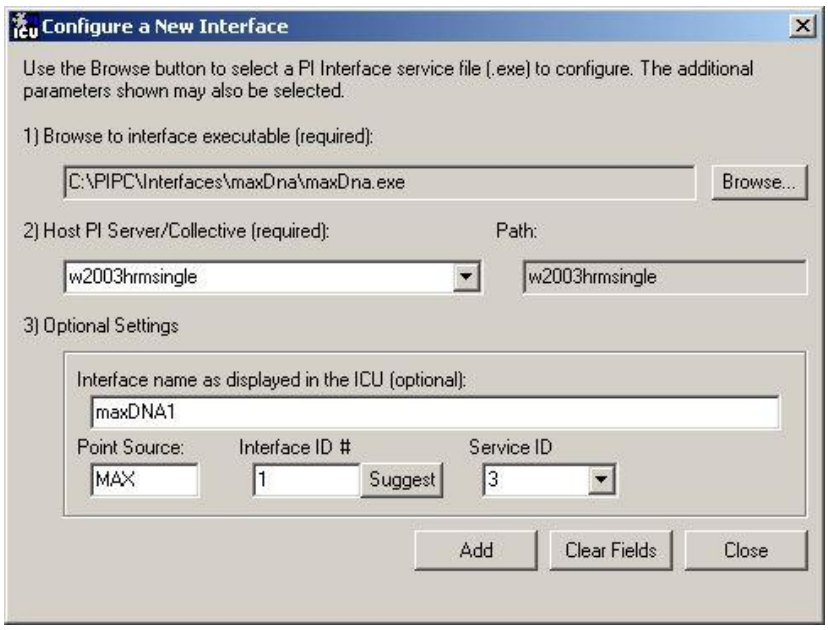
The PI Interface Configuration Utility (PI ICU) provides a tool for configuring the interface startup command file.

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 (`PIMax.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 maxDNA interface.

From the PI ICU menu, select *Interface*, then *NewWindows Interface Instance from EXE...*, and then *Browse* to the `PIMax.exe` executable file. Then, enter values for *Host PI System*, *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.

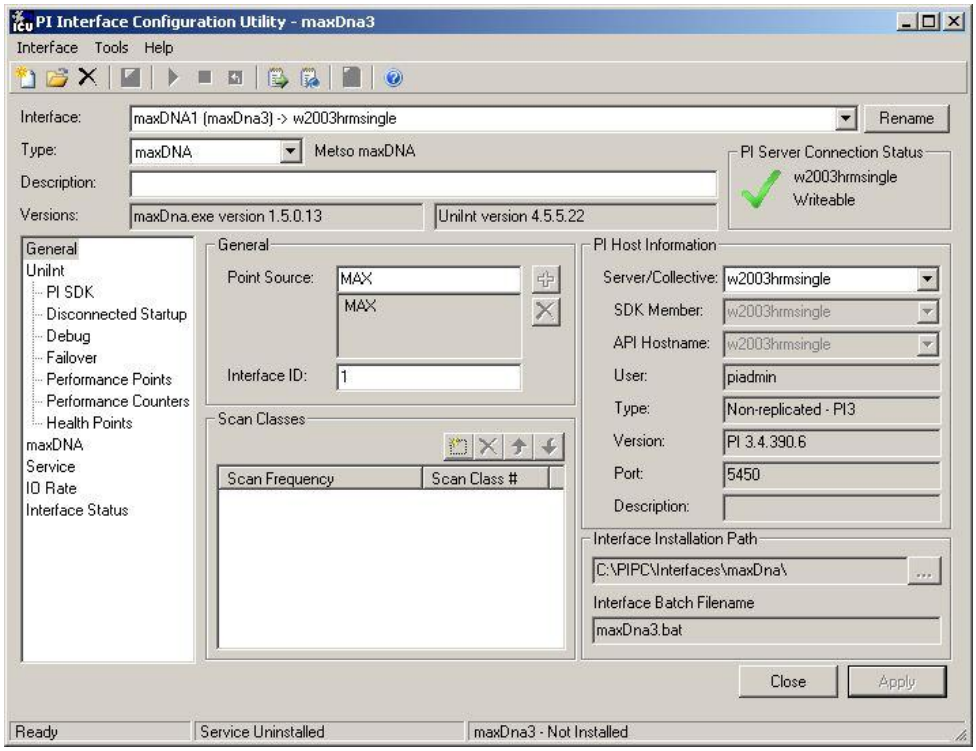
Click *Add*.

The following message should appear:

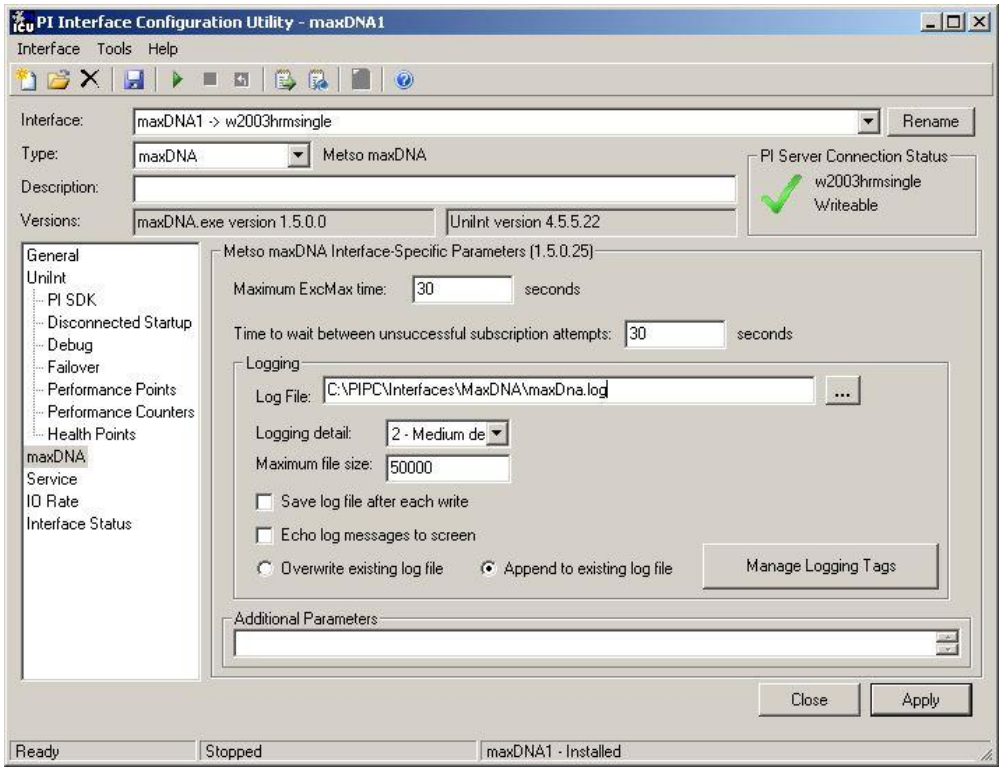
Note that in this example the Host PI Server is w2003hrmsingle. To configure the interface to communicate with a remote PI Server, select *Connections…* from the PI ICU *Interface* menu and select the default server. If the remote node is not present in the list of servers, it can be added.

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

Click on *Apply* to enable the PI ICU to manage this instance of the maxDNA interface.



The next step is to make selections in the interface-specific page (that is, "`maxDNA`") that allows you to enter values for the startup parameters that are particular to the maxDNA interface.



Since the maxDNA 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.

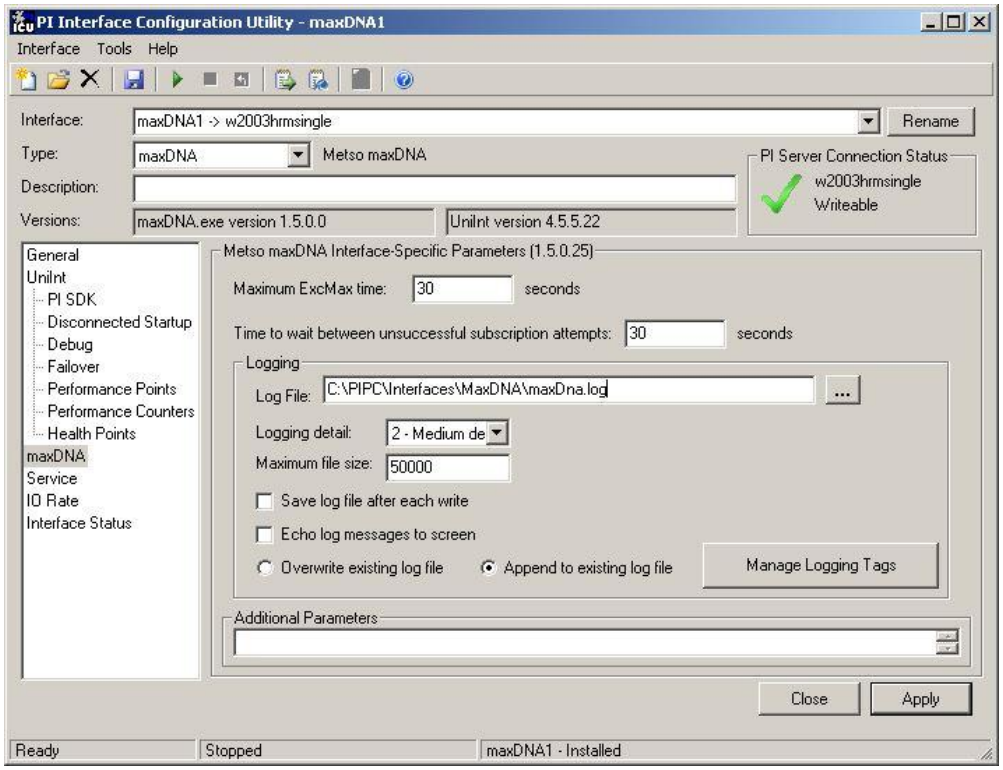
To set up the interface as a Windows Service, use the *Service* page. This page allows configuration of the interface to run as a service as well as to starting and stopping of the interface service. The interface can also be run interactively from the PI ICU. To do that, select *Start Interactive* on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

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 guide. The next section describes the selections that are available from the *maxDNA* 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.

maxDNA Interface Page

Since the startup file of the maxDNA interface is maintained automatically by the PI ICU, use the *maxDNA* 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.

maxDNA



The PI Interface for Metso maxDna – ICU Control has 2 sections. A yellow text box indicates that an invalid value has been entered or that a required value has not been entered.

Maximum ExcMax Time

This field specifies the maximum exception time, which is the maximum value that a tag's excmax attribute can be set to in seconds. If the **`/excmax`** parameter is not set, the maximum exception maximum time is 30 seconds. Consult your Metso Automation representative for recommendations for exception maximum settings, usually this value should be 30.

Time to Wait Between Unsuccessful Subscription Attempts

This field specifies the amount of time (in seconds) the interface should wait between unsuccessful subscription attempts. Setting this too low will cause unnecessary network traffic and will use more processor time

Logging section

Log File

This field is used to specify the name and location of the logging file. Clicking the **Browse** button displays a dialog box that you can use to browse to an existing or create a new logging file.

Logging detail

This combo box is used to configure the detail in logging occurs. The available selections are:

- Low
- Medium
- High
- None
- Maximum

Maximum file size

This field is used to control the maximum size, in bytes, that the logging file is allow to grow to.

Save log file after each write

If this option is selected, the interface will commit each log message to disc as it is created. If the logging level is set to High or Maximum this may create substantial overhead on the system so it is suggested to be used with care.

Echo log messages to screen

Select this option to cause the interface to echo all logging messages to the screen. The interface must be run interactively in order for this to work.

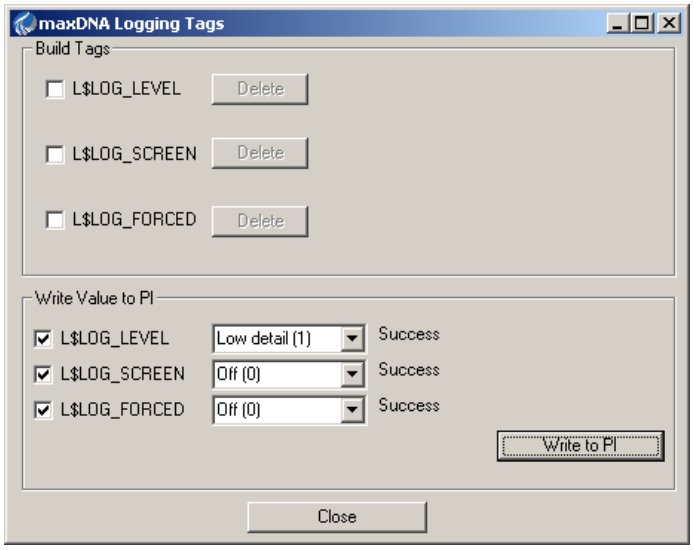
Overwrite / Append to existing log file

This field is used to specify the overwrite / append options for the logging file. If Overwrite is selected a new log file is created each time the interface starts. If append is selected, the interface will append any new message to the previous log file.

Manage Logging Tags

You can change the operation of the logging facilities at runtime by creating several Logging Tags. The logging tags provide a mechanism for changing the logging configuration during run-time. When the logging configuration needs to be changed, a value can be written to the appropriate tag. For each possible logging configuration change, there is a specific PI tag.

This screen allows the runtime logging tags to be created, deleted and modified.



Build Tags

This section allows the logging control tags to be created if they do not already exist or deleted if they do exist. Select the desired tag and then click **Build** or **Delete** next to the tag.

Write Values to PI

This section allows the logging options to be controlled via the logging control tags. Select the logging tag to be modified and the select the desired value via the combo box and then click the Write to PI button on the right.

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
/CacheMode Required when using disconnected startup 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 Server. 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)
/D=# Optional Default: 2	The **/D** parameters specify the detail level for logging. Medium detail logging is set by default. Supported values: 9 = All logs 8 = No logs 3 = High detail 2 = Medium detail 1 = Low detail

Parameter	Description
/ec=*#* Optional	The first instance of the **/ec** parameter on the command-line is used to specify a counter number, *#*, for an I/O Rate point. If the *#* is not specified, then the default event counter is 1. Also, if the **/ec** parameter is not specified at all, there is still a default event counter of 1 associated with the interface. If there is an I/O Rate point that is associated with an event counter of 1, every interface that is running without **/ec=***#* explicitly defined will write to the same I/O Rate point. Either explicitly define an event counter other than 1 for each instance of the interface or do not associate any I/O Rate points with event counter 1. Configuration of I/O Rate points is discussed in the section called I/O Rate Point. For interfaces that run on Windows nodes, subsequent instances of the **/ec** parameter may be used by specific interfaces to keep track of various input or output operations. Subsequent instances of the **/ec** parameter can be of the form **/ec***, where * is any ASCII character sequence. For example, **/ecinput=10**, **/ecoutput=11**, and **/ec=12** are legitimate choices for the second, third, and fourth event counter strings.
/excmax=# default: 30 seconds	Specifies the maximum exception time, which is the maximum value that a tag's excmax attribute can be set to in seconds. If the **/excmax** parameter is not set, the maximum exception maximum time is 30 seconds. Consult your Metso Automation representative for recommendations for exception maximum settings, usually this value should be 30.
/f=*SS.##* or **/f=***SS.##,ss.##* or **/f=***HH:MM:SS.##* or **/f=***HH:MM:SS.##,* *hh:mm:ss.##* Required for reading scan-based inputs	The **/f** parameter defines the time period between scans in terms of hours (*HH*), minutes (*MM*), seconds (*SS*) and sub-seconds (*##*). The scans can be scheduled to occur at discrete moments in time with an optional time offset specified in terms of hours (*hh*), minutes (*mm*), seconds (*ss*), and sub-seconds (*##*). If *HH* and *MM* are omitted, then the time period that is specified is assumed to be in seconds. Each instance of the **/f** parameter on the command-line defines a scan class for the interface. There is no limit to the number of scan classes that can be defined. The first occurrence of the **/f** parameter on the command-line defines the first scan class of the interface; the second occurrence defines the second scan class, and so on. PI Points are associated with a particular scan class via the Location4 PI Point attribute. For example, all PI Points that have Location4 set to 1 will receive input values at the frequency defined by the first scan class. Similarly, all points that have Location4 set to 2 will receive input values at the frequency specified by the second scan class, and so on. Two scan classes are defined in the following example: **/f=00:01:00,00:00:05 /f=00:00:07** or, equivalently: **/f=60,5 /f=7** The first scan class has a scanning frequency of 1 minute with an offset of 5 seconds, and the second scan class has a scanning frequency of 7 seconds. When an offset is specified, the scans occur at discrete moments in time according to the formula: scan times = (reference time) + n(frequency) + offset where n is an integer and the reference time is midnight on the day that the interface was started. In the above example, frequency is 60 seconds and offset is 5 seconds for the first scan class. This means that if the interface was started at 05:06:06, the first scan would be at **05:07:05**, the second scan would be at **05:08:05**, and so on. Since no offset is specified for the second scan class, the

Parameter	Description

	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 *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.

`/force` Optional	The **/force** parameter enabling forced writes to the log file. When enabled commits all data to the drive after each write (slows the process significantly).
`/host=`*`host:port`* Required	The **`/host`** parameter is used to specify the PI Home node. *Host* is the IP address of the PI Server node or the domain name of the PI Server node. *Port* is the port number for TCP/IP communication. The *port* is always 5450. It is recommended to explicitly define the host and port on the command-line with the **`/host`** parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults. Examples: The interface is running on a 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
/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 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 instance 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**
/L=xxx Optional	The **/L** specifies the file for logging. If the parameter is not specified, the default log file is used: "C:\Users\<USERNAME>\AppData\Roaming\MaxDna.log"
/M=# Optional Default: 50000	The **/M** specifies the max log file size in bytes. When the log file size reach the limit older messages are overwritten.
/N=x Optional Default:1(True)	The **/N** specify how the interface write to the log form the start. When the parameter is 1, the new log file is created. When the parameter is 0 log messages are appended to the current file.
/ps=*x* Required	The **/ps** parameter specifies the point source for the interface. *X* is not case sensitive and can be any single/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 Server version is prior to 3.4.370.x, the PointSource is limited to a single character unless the SDK is being used.
/s Optional	The **/s** parameter enable screen logging.
/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.

Parameter	Description
/stopstat=*digstate* or **/stopstat** **/stopstat** only is equivalent to **/stopstat="***Intf Shut***"** Optional Default = no digital state written at shutdown.	If **/stopstat=***digstate* is present on the command line, then the digital state, *digstate,* will be written to each PI point when the interface is stopped. For a PI3 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 chapter 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 must be enclosed within double quotes when there is a space in *digstate*.
/subchk=x default: 30 seconds	Specifies the amount of time (in seconds) the interface should wait between unsuccessful subscription attempts. Setting this too low will cause unnecessary network traffic and will use more processor time.
/UFO_ID=*#* Required for UniInt Failover Phase 1 or 2	Failover ID. This value must be different from the Failover ID of the other interface in the failover pair. It can be any positive, non-zero integer.
/UFO_Interval=*#* Optional Default: `1000 for` Phase 1 Failover Default: `5000` for Phase 2 Failover Valid values are 50-20000.	Failover Update Interval Specifies the heartbeat Update Interval in milliseconds and must be the same on both interface computers. This is the rate at which UniInt updates the Failover Heartbeat tags as well as how often UniInt checks on the status of the other copy of the interface.
/UFO_OtherID=*#* Required for UniInt Failover Phase 1 or 2	Other Failover ID. This value must be equal to the Failover ID configured for the other interface in the failover pair.

Parameter	Description
/UFO_Sync=*path/[filename]* Required for UniInt Failover Phase 2 synchronization. Any valid pathname / any valid filename The default filename is generated as *executablename_pointsource_interfaceID.dat*	The Failover File Synchronization file *path* and optional *filename* specify the path to the shared file used for failover synchronization and an optional filename used to specify a user defined filename in lieu of the default filename. The *path* to the shared file directory can be a fully qualified machine name and directory, a mapped drive letter, or a local path if the shared file is on one of the interface nodes. The *path* must be terminated by a slash (/) or backslash (\ ) character. If no d terminating slash is found, in the **/UFO_Sync** parameter, the interface interprets the final character string as an optional *filename*. The optional *filename* can be any valid filename. If the file does not exist, the first interface to start attempts to create the file. **Note:** If using the optional filename, **do not** supply a terminating slash or backslash character. If there are any spaces in the *path* or *filename*, the entire path and filename must be enclosed in quotes. **Note:** If you use the backslash and path separators and enclose the path in double quotes, the final backslash must be a double backslash (\\). Otherwise the closing double quote becomes part of the parameter instead of a parameter separator. Each node in the failover configuration must specify the same path and filename and must have read, write, and file creation rights to the shared directory specified by the *path* parameter. The service that the interface runs against must specify a valid logon user account under the “Log On” tab for the service properties.
/UFO_Type=*type* Required for UniInt Failover Phase 2.	The Failover Type indicates which type of failover configuration the interface will run. The valid types for failover are HOT, WARM, and COLD configurations. If an interface does not supported the requested type of failover, the interface will shut down and log an error to the `pipc.log` file stating the requested failover type is not supported.
/uht_id=*#* Optional Required if any type of failover other than UniInt Failover Phase 1 or 2 is supported.	The **/uht_id=***#* command-line parameter is used to specify a unique ID for interfaces that are run in a redundant mode without using the UniInt failover mechanism. There are several OSIsoft interfaces that are UniInt based and implement their own version of failover. In order for health tag(s) to be configured to monitor a single copy of the Interface, an additional parameter is required. If the **/uht_id=***#* is specified; only health tags with a Location3 value equal to # will be loaded.

Sample PIMax.bat File

The following is an example file:

```
REM=================================================================
REM
REM PIMax.bat
REM
REM Sample startup file for the PI Interface for Metso maxDna
REM
REM=================================================================
REM
REM OSIsoft strongly recommends using PI ICU to modify startup files.
REM
REM Sample command line
REM

    .\PIMax.exe ^
    /PS=M ^
    /host=localhost:5450 ^
    /ID=1 ^
    /maxstoptime=120 ^
    /sio
REM
REM End of PIMax.bat File
```

Chapter 9. UniInt Failover Configuration

Introduction

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

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

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

Note: This interface supports only Phase 2 failover.

You can also configure UniInt failover to send data to a High Availability (HA) PI Server collective. The collective provides redundant PI Servers to allow for the uninterrupted collection and presentation of PI time series data. In an HA configuration, PI Servers can be taken down for maintenance or repair. The HA PI Server collective is described in the *High Availability Administrator 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 chapter 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 chapter in detail. If a failure occurs at any step below, correct the error and start again at the beginning of step 6 Test in the table below. For the discussion below, the first copy of the interface configured and tested will be considered the primary interface and the second copy of the interface configured will be the backup interface.

Configuration

- One Data Source
- Two Interfaces

Prerequisites

- Interface 1 is the primary interface for collection of PI data from the data source.
- Interface 2 is the backup interface for collection of PI data from the data source.
- You must setup 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 Server host for exceptions and PI tag updates.
- All other configuration parameters for the two interfaces must be identical.

Synchronization through a Shared File (Phase 2)

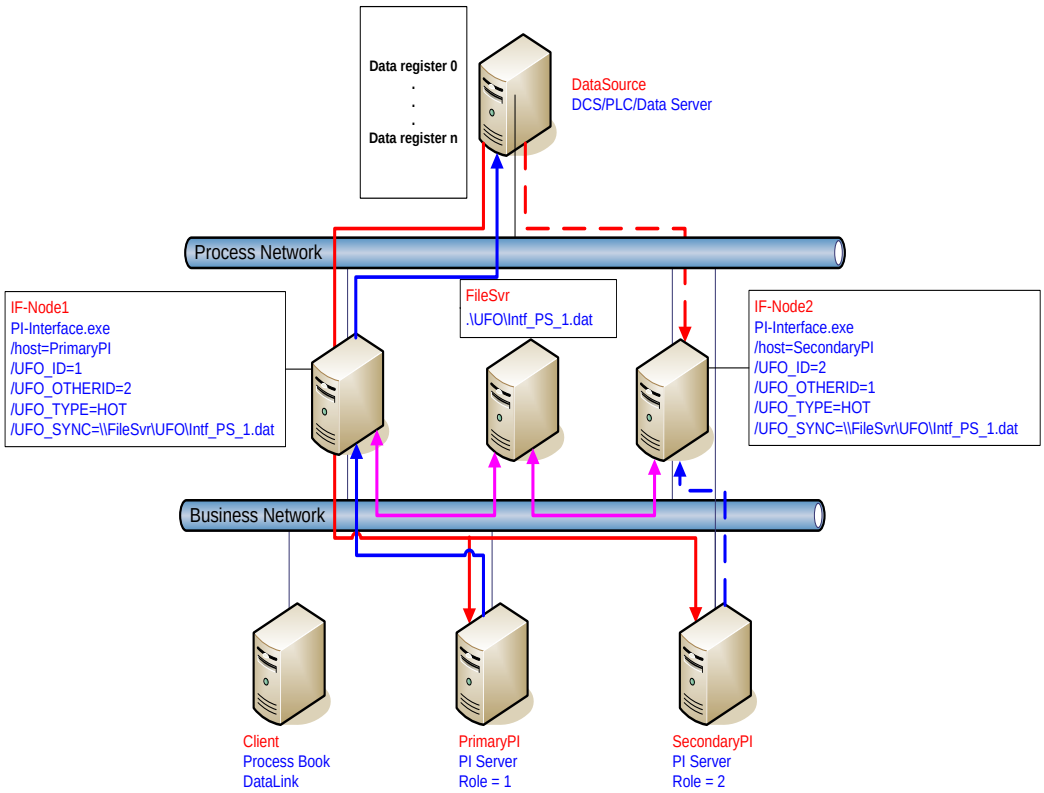


Figure : Synchronization through a Shared File (Phase 2) Failover Architecture

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

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

Configuring Synchronization through a Shared File (Phase 2)

<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. .
Setup a file share and make sure to assign the permissions so that both primary and backup interfaces have read/write access to the file.</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>
</table>

Step	Description
5.	**Test the configuration.** 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 Server using available PI tools. • The **Active ID** control tag on the PI Server must be set to the value of the running copy of the interface as defined by the **`/UFO_ID`** startup command-line parameter. • The **Heartbeat** control tag on the PI Server must be changing values at a rate specified by the **`/UFO_Interval`** startup command-line parameter. 4. Stop the primary interface. 5. Start the backup interface interactively without buffering. Notice that this copy will become the primary because the other copy is stopped. 6. Repeat steps 2, 3, and 4. 7. Stop the backup interface. 8. Start buffering. 9. Start the primary interface interactively. 10. Once the primary interface has successfully started and is collecting data, start the backup interface interactively. 11. Verify that both copies of the interface are running in a failover configuration. • Review the `pipc.log` file for the copy of the interface that was started first. The log file will contain messages that indicate the failover state of the interface. The state of this interface must have changed as indicated with an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 Server using available PI tools. • The **Active ID** control tag on the PI 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 tags for both copies of the interface on the PI

Step	Description
	Server must be changing values at a rate specified by the **/UFO_Interval** startup command-line parameter or the scan class which the points have been built against. 13. Test Failover by stopping the primary interface. 14. Verify the backup interface has assumed the role of primary by searching the `pipc.log` file for a message indicating the backup interface has changed to the "`UniInt failover: Interface in the "Primary" state and actively sending data to 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 file *path* and optional *filename* specify the path to the shared file used for failover synchronization and an optional filename used to specify a user defined filename in lieu of the default filename. The *path* to the shared file directory can be a fully qualified machine name and directory, a mapped drive letter, or a local path if the shared file is on one of the interface nodes. The *path* must be terminated by a slash (/) or backslash (\ ) character. If no terminating slash is found, in the **/UFO_Sync** parameter, the interface interprets the final character string as an optional *filename*. The optional *filename* can be any valid filename. If the file does not	Any valid pathname / any valid filename The default filename is generated as *executablename_ pointsource_ interfaceID.dat*

Parameter	Required/ Optional	Description	Value/Default
		exist, the first interface to start attempts to create the file. **Note:** If using the optional filename, **do not** supply a terminating slash or backslash character. If there are any spaces in the *path* or *filename*, the entire path and filename must be enclosed in quotes. **Note:** If you use the backslash and path separators and enclose the path in double quotes, the final backslash must be a double backslash (`\\`). Otherwise the closing double quote becomes part of the parameter instead of a parameter separator. Each node in the failover configuration must specify the same path and filename and must have read, write, and file creation rights to the shared directory specified by the *path* parameter. The service that the interface runs against must specify a valid logon user account under the "Log On" tab for the service properties.	
/UFO_Type=*type*	Required	The Failover Type indicates which type of failover configuration the interface will run. The valid types for failover are HOT, WARM, and COLD configurations. If an interface does not supported the requested type of failover, the interface will shutdown and log an error to the `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 Server for exceptions and PI point updates The value of the **/Host** startup parameter depends on the PI Server configuration. If the PI Server is not part of a collective, the value of **/Host** must be identical on both interface computers. If the redundant interfaces are being configured to send data to a PI Server collective, the value of the **/Host** parameters on the different interface nodes should equal to different members of the collective. This parameter ensures that outputs continue to be sent to the data source if one of the PI Servers becomes unavailable for any reason.	For IF-Node1 PrimaryPI / **None** For IF-Node2 SecondaryPI / **None**

Failover Control Points

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

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.
PointSource	Match *x* in **/ps=***x*
PointType	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*
PointType	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

Attribute	Primary	Backup
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
[UFO2_ACTIVEID]	Required	Active ID tag The ExDesc must start with the case sensitive string: [UFO2_ACTIVEID]. The PointSource must match the interfaces' Pointsource. 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' PointSource. 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' PointSource. 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' PointSource. Location1 must match the ID for the interfaces. A lower value is a better status and the interface with the lower status will attempt to become the primary interface.	0 – 99 / **None** Updated by the interface on IF-Node2
[UFO2_STATE:#] (IF-Node1)	Optional	State 1 Tag The ExDesc must start with the case sensitive string: [UFO2_STATE:#] The number following the colon (:) must be the Failover ID for the interface running on IF-Node1 The failover state tag is recommended. The failover state tags are digital tags assigned to a digital state set with the following values. 0 = Off: The interface has been shut down. 1 = Backup No Data Source: The	0 – 5 / **None** Normally updated by the interface currently in the primary role.

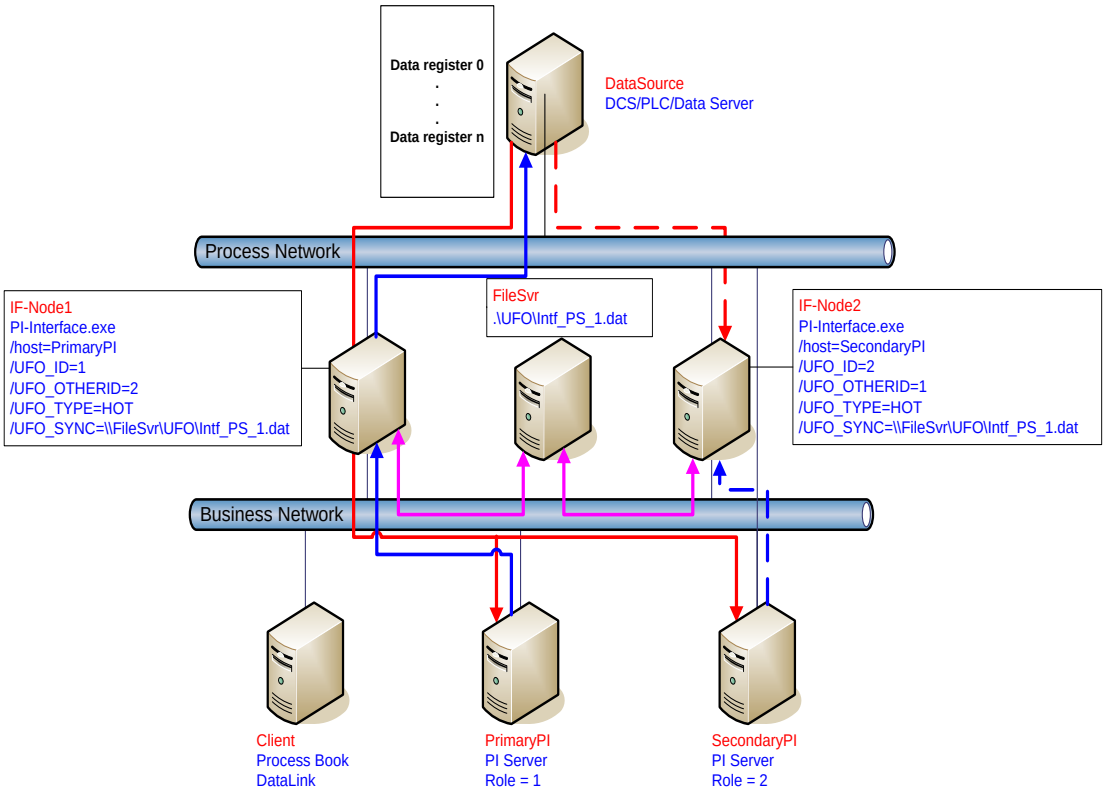
PI Tag ExDesc	Required / Optional	Description	Value
		interface is running but cannot communicate with the data source. 2 = Backup No PI Connection: The interface is running and connected to the data source but has lost its communication to the PI Server. 3 = Backup: The interface is running and collecting data normally and is ready to take over as primary if the primary interface shuts down or experiences problems. 4 = Transition: The interface stays in this state for only a short period of time. The transition period prevents thrashing when more than one interface attempts to assume the role of primary interface. 5 = Primary: The interface is running, collecting data and sending the data to 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 Server 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 Server 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 Server as an alternate path to pass control data.

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

The primary interface also monitors the connection with the shared file to maintain the integrity of the failover configuration. If the primary interface can read and write to the shared file with no errors but the backup control information is not changing, then the backup is experiencing some error condition. To determine exactly where the problem exists, the primary interface uses the path to 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 Server. Updating of the tags on the PI Server is assumed to take place unless communication with the PI Server is interrupted. The updates to the PI Server 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 Server is lost, the value of the heartbeat will be incremented from 17 – 31 and then wrap around to a value of 17 again. Once the connection to the PI Server is restored, the heartbeat values will revert back to the 1 – 15 range. During a normal shutdown process, the heartbeat value will be set to zero.

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

As previously mentioned, in a hot failover configuration the backup interface actively collects data but does not send its data to 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 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 #* boxes will be in yellow and a message will be displayed saying this is already in use. This should be ignored.

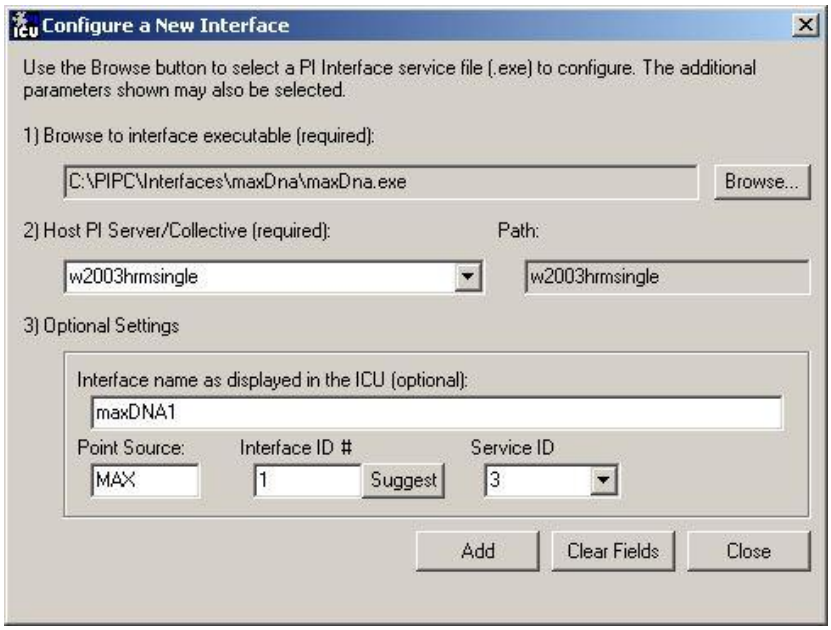


Figure : PI ICU configuration screen shows that the "PS/ID combo is already in use by the interface." The user must ignore the yellow boxes, which indicate errors, and click the *Add* button to configure the interface for failover.

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

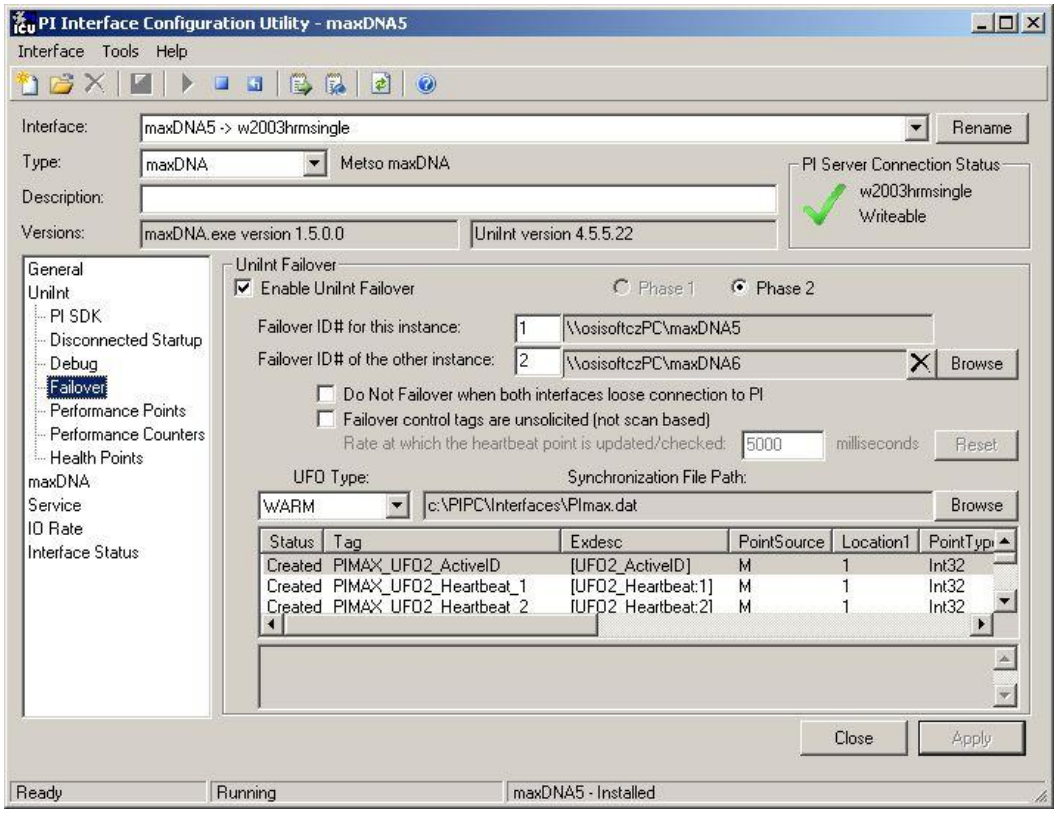


Figure : The figure above illustrates the PI ICU failover configuration screen showing the UniInt failover startup parameters (Phase 2). This copy of the interface defines its Failover ID as 2 (**/UFO_ID=2**) and the other Interfaces Failover ID as 1 (**/UFO_OtherID=1**). The other failover interface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

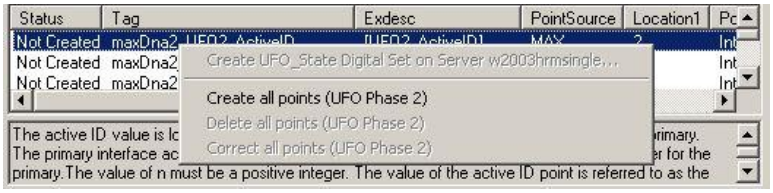
Creating the Failover State Digital State Set

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 created 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 click the *Create UFO_State Digital Set on Server*

XXXXXX... command, where *XXXXXX* is the PI Server where the points will be or are created.



This command will be unavailable if the UFO_State digital state set already exists on the *XXXXXX* PI Server.

Using the PI SMT 3 Utility to create Digital State Set

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

The procedure below outlines the steps necessary to create a digital set on a PI Server using the *Import from File* command 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 Server 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 below.
3. From the *System Management Plug-Ins* window, expand *Points* then select *Digital States*. A list of available digital state sets will be displayed in the main window for the selected PI Server. Refer to Figure below.
4. In the main window, right-click on the desired server and select the *Import from File* command. Refer to Figure below.

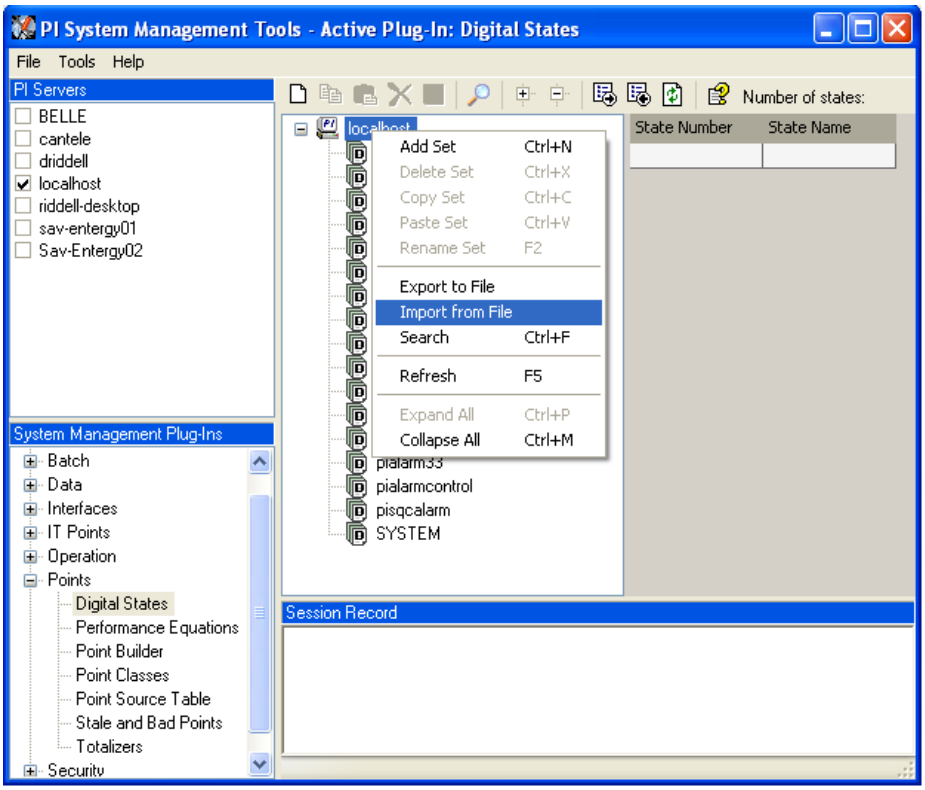


Figure : PI SMT application configured to import a digital state set file. The PI Servers window shows the "localhost" PI Server selected along with the System Management Plug-Ins window showing the Digital States Plug-In as being selected. The digital state set file can now be imported by selecting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

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

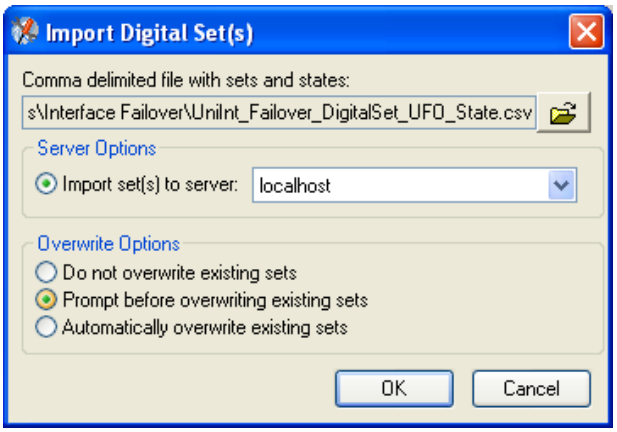


Figure : PI SMT application *Import Digital Set(s)* window. This view shows the `UniInt_Failover_DigitalSet_UFO_State.csv` file as being selected for import. Select the desired *Overwrite Options* by choosing the appropriate option button.

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

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

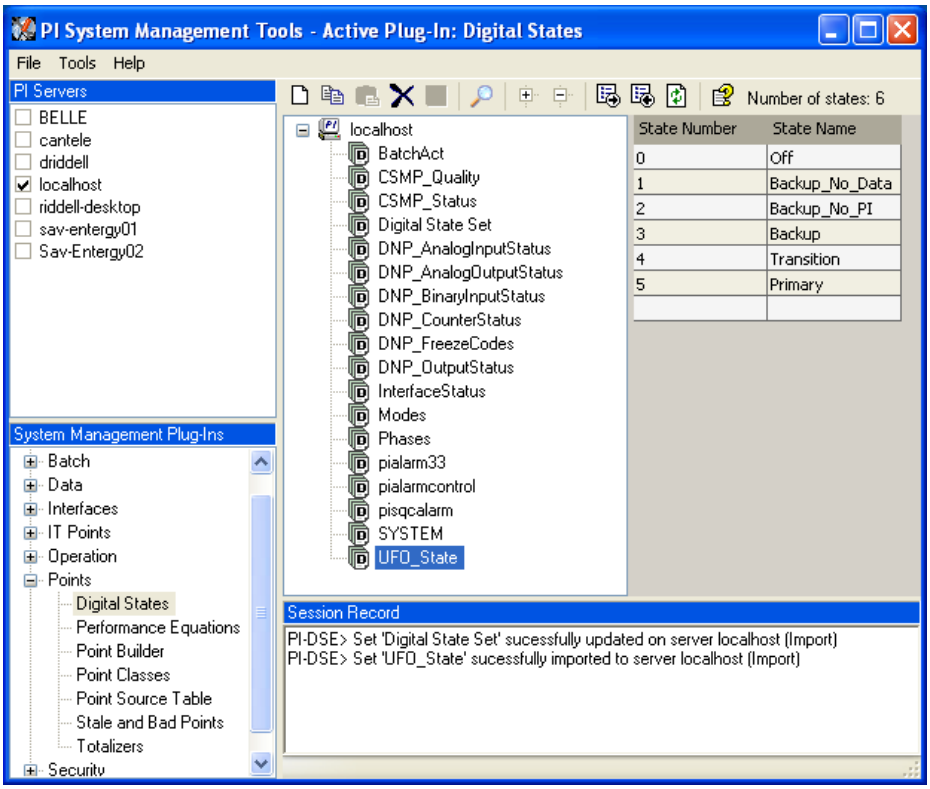


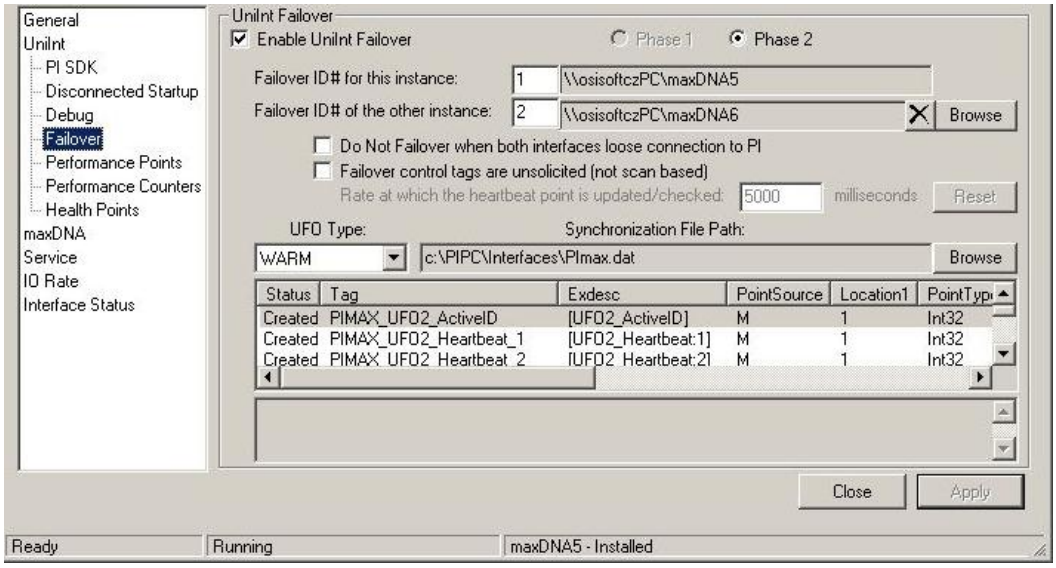
Figure : The PI SMT application showing the `UFO_State` digital set created on the "localhost" PI Server.

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

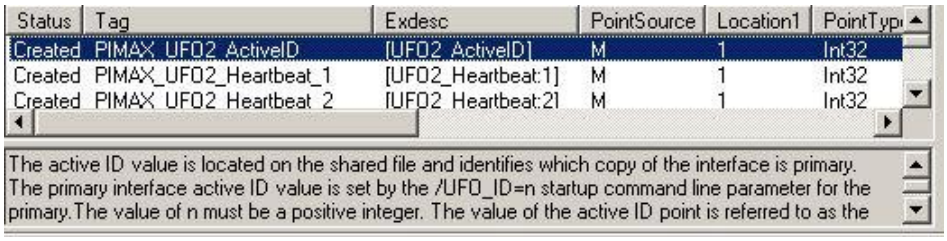
The ICU can be used to create the UniInt Failover Control and State Tags.

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

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



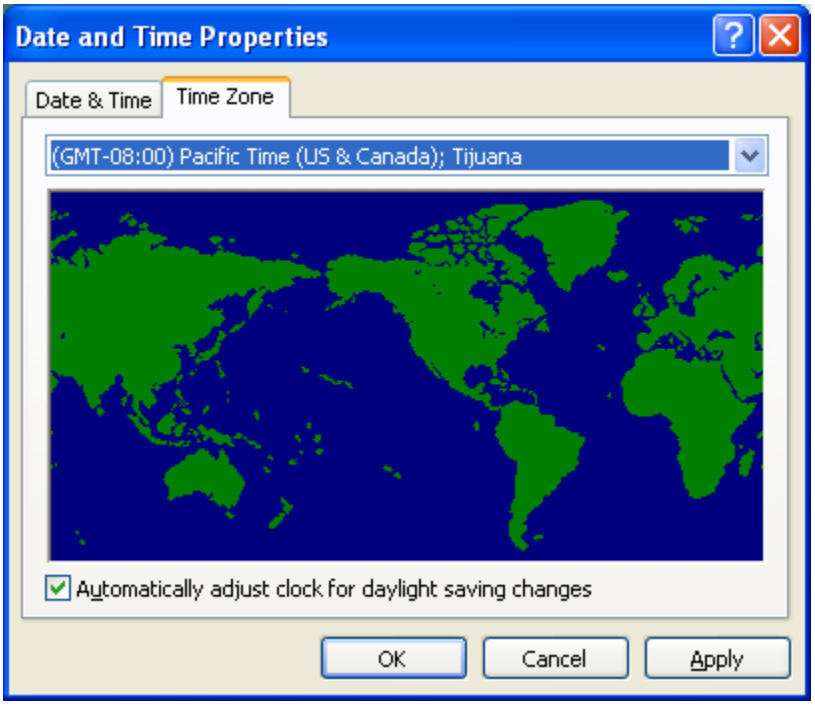
Once the failover control and failover state tags have been created the *Failover* page of the ICU should look similar to the illustration below.



Chapter 10. Interface Node Clock

Windows

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 11. Security

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

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

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

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

PI Server v3.3 and Higher

Security configuration using piconfig

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

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

For the above,

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

```
a_trust_name
```

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

```
192.168.100.11
```

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

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

Security Configuring using Trust Editor

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

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

PI Server v3.2

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

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

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

Chapter 12. Starting / Stopping the Interface on Windows

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:

```
PIMax.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:

```
PIMax.exe /stop
```

The service can be removed by:

```
PIMax.exe /remove
```

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

Chapter 13. Buffering

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

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

If you have PI Servers 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 Servers 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 Servers that are part of that collective.

You can use PIBufss only under the following conditions:

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

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

- the PI Server version is earlier than 3.4.375.x; or
- the interface node runs multiple interfaces, and these interfaces send data to multiple PI Servers 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 Server 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 Server. It also creates shared memory storage.

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

The buffering application (either Bufserv or PIBufss) in turn

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

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

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

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

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

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

When buffering is enabled, it affects the entire interface node. That is, you do not have a 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 Server Security

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

However, if the PI Server 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:

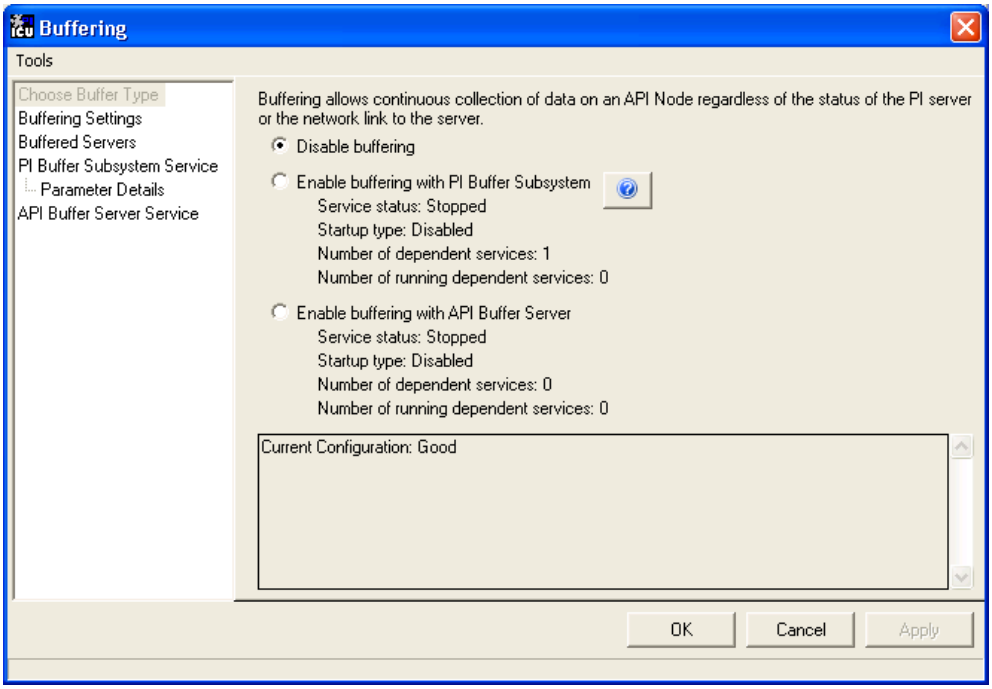
Buffering Application	Application Name field for PI Trust
PI Buffer Subsystem	PIBufss.exe
PI API Buffer Server	APIBE (if the PI API is using 4 character process names) APIBUF (if the PI API is using 8 character process names)

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*.

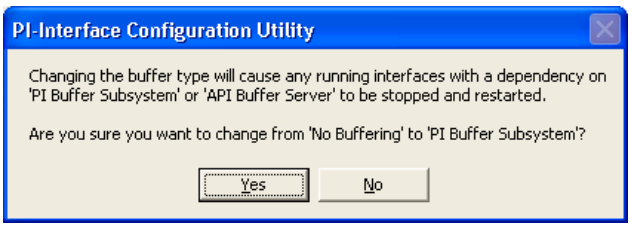
Choose Buffer Type



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*.

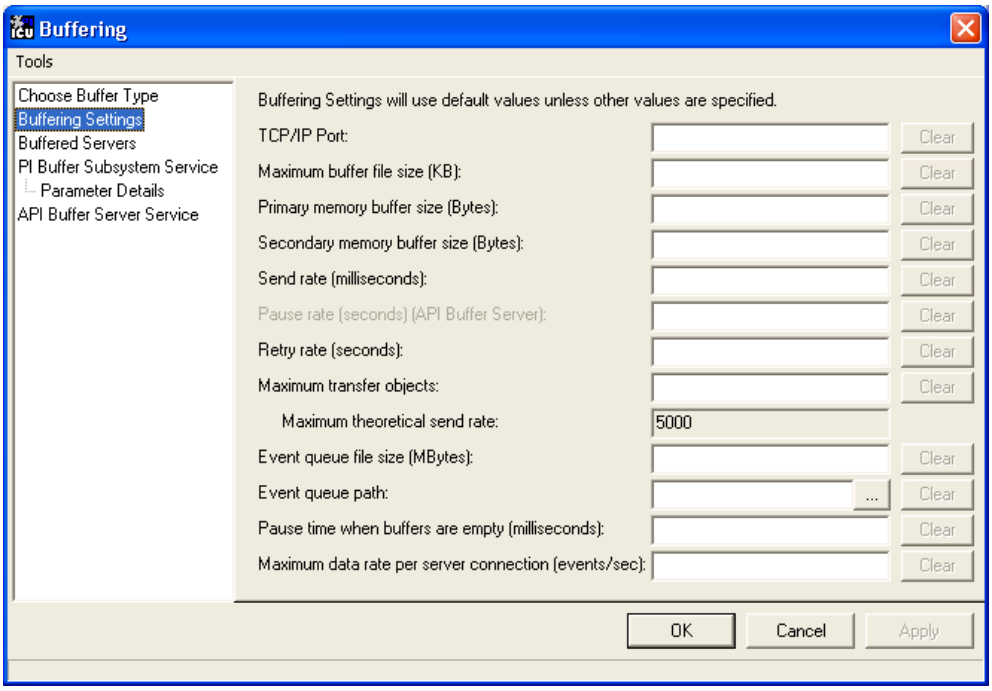


Buffering Settings

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 Server. The default value is 100. The valid range is 0 to 2,000,000.

Maximum transfer objects

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

Event Queue File Size (Mbytes)

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

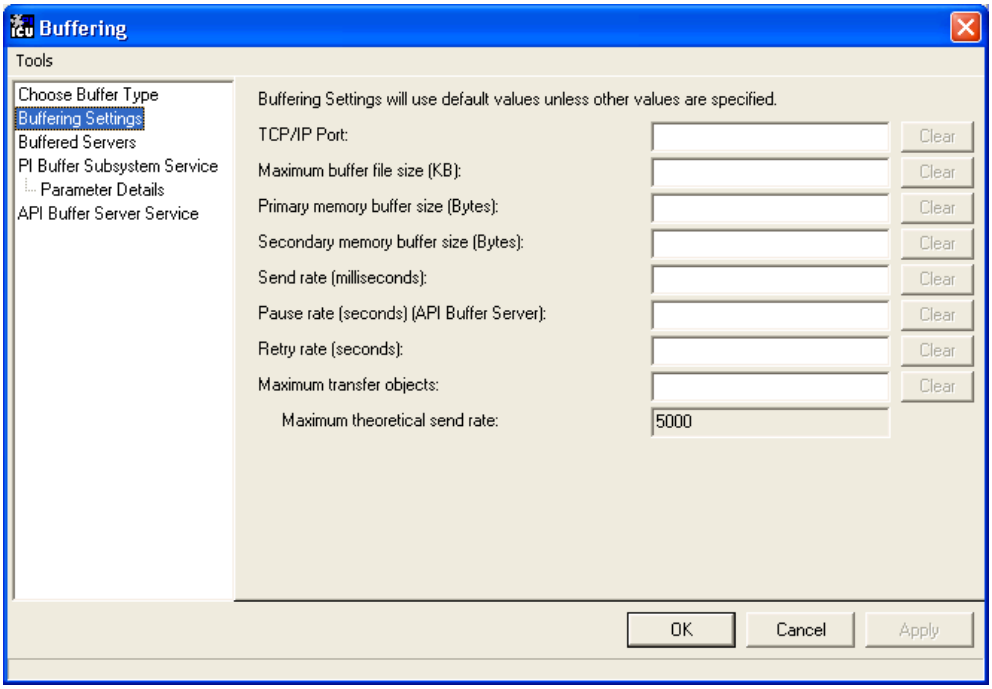
Event Queue Path

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

For optimal performance and reliability, 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 Server, it writes and appends data to this file. When the buffer file reaches this maximum size, Bufserv discards data.

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

Primary and Secondary Memory Buffer Size (Bytes)

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

The default value of these memory buffers is 32,768 bytes. 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 Server. 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.

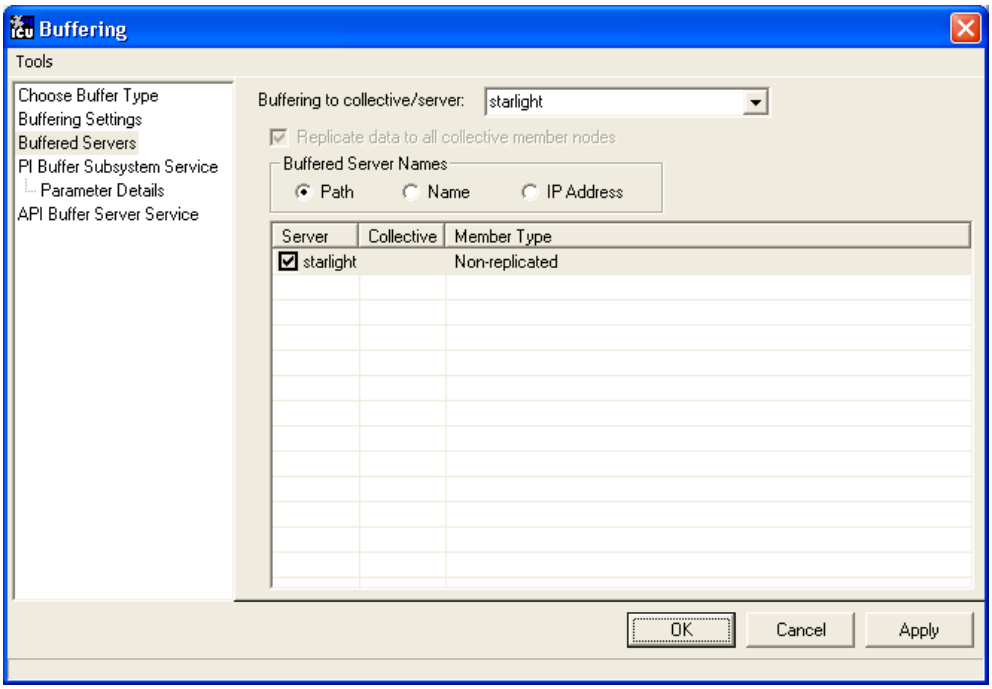
Buffered Servers

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

PIBufss

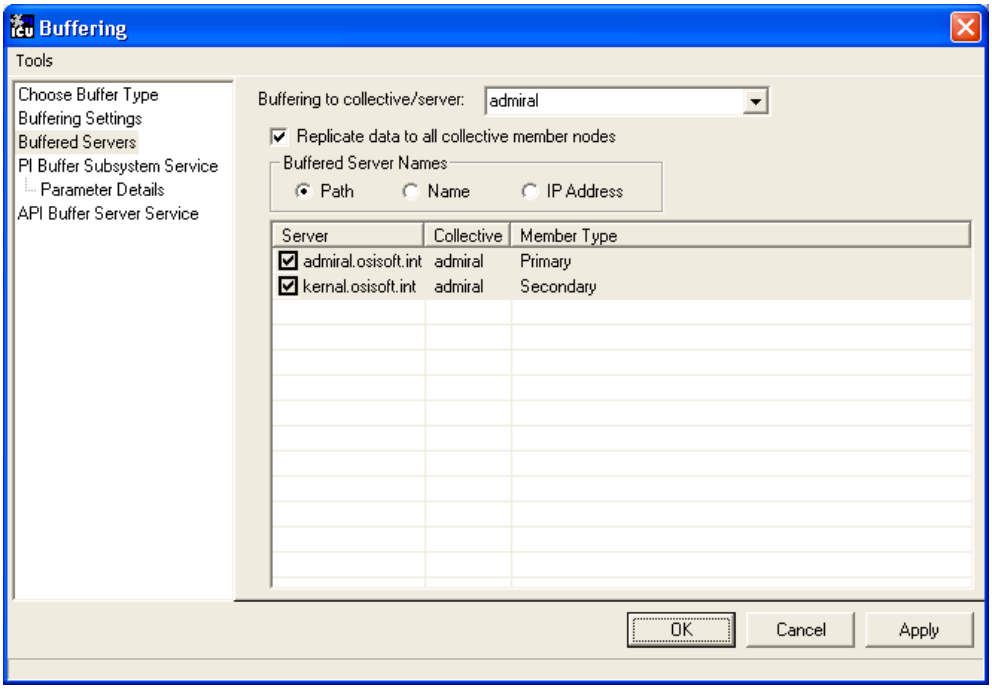
PIBufss buffers data only to a single PI Server or a PI collective. Select the PI Server or the PI collective from the *Buffering to collective/server* drop down list box.

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

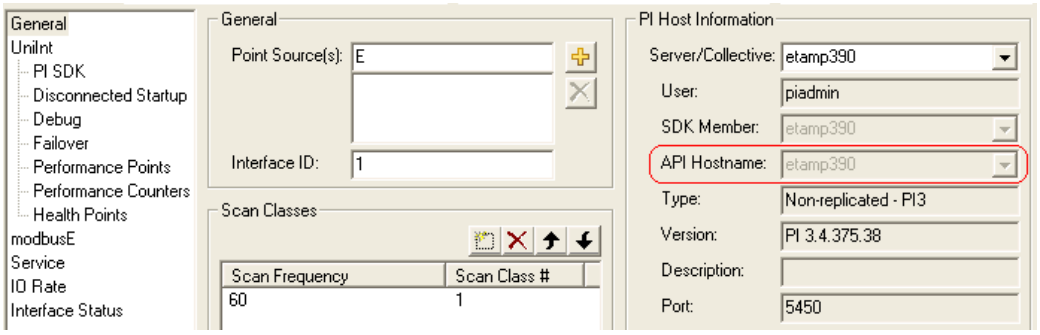
You can override this option by not checking the *Replicate data to all collective member nodes* check box. Then, uncheck (or check) the PI Server collective members as desired.



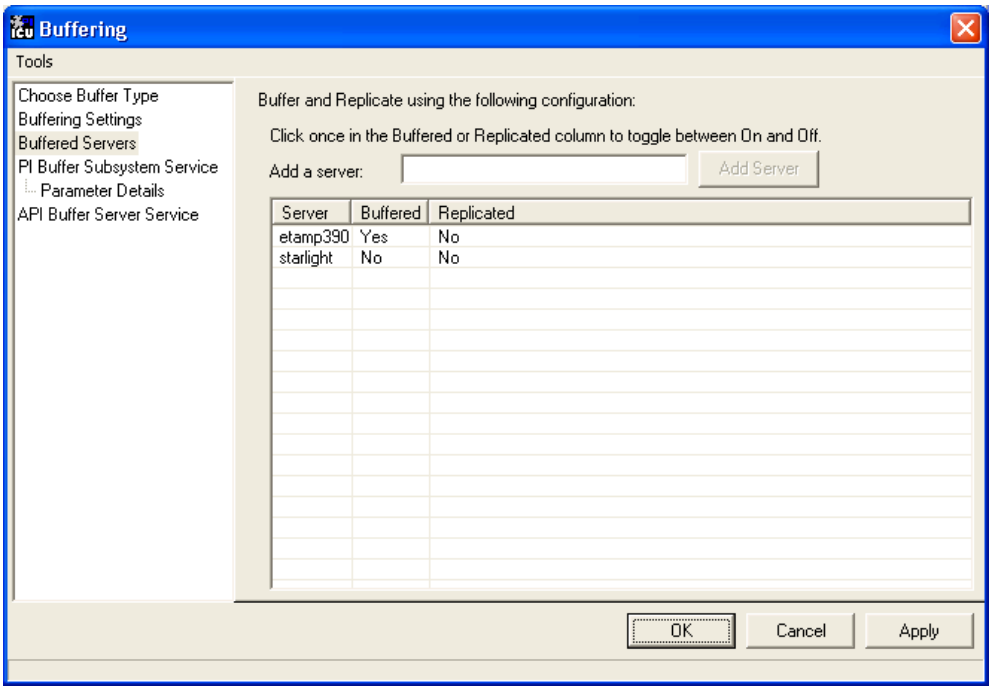
Bufserv

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

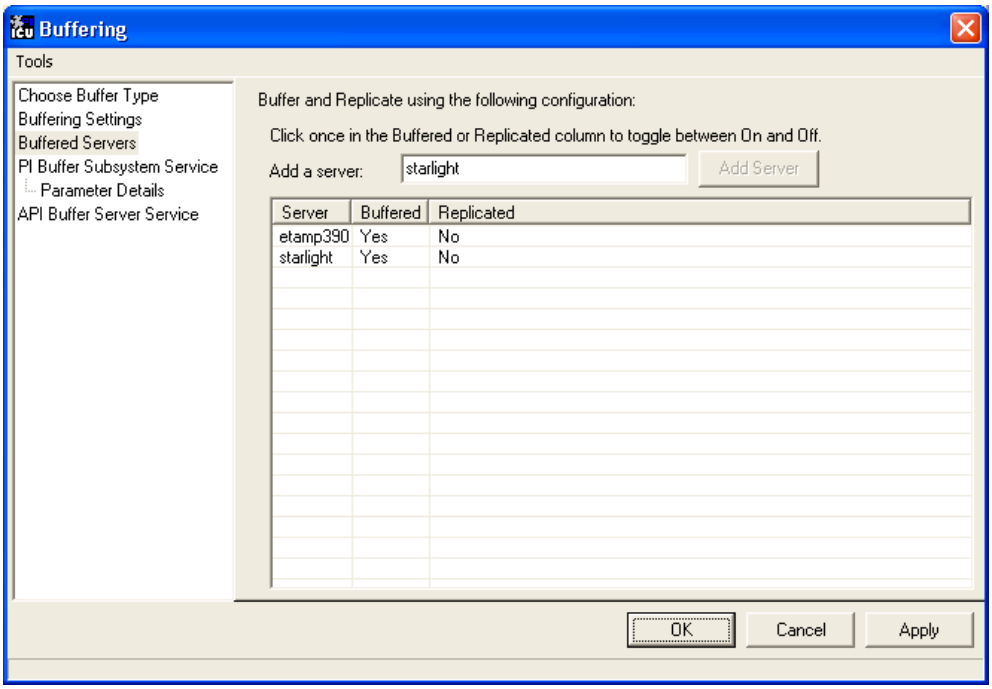
If the PI Server to which you want Bufserv to buffer data is not in the Server list, enter its name in the *Add a server* box and click the *Add Server* button. This PI Server name must be identical to the **API Hostname** entry:



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



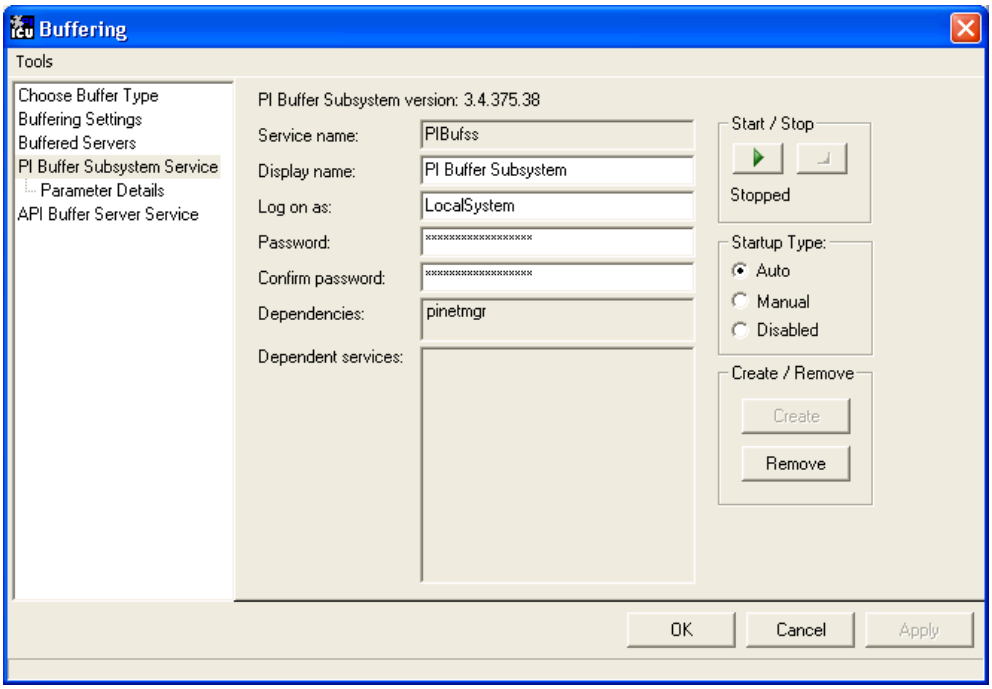
Installing Buffering as a Service

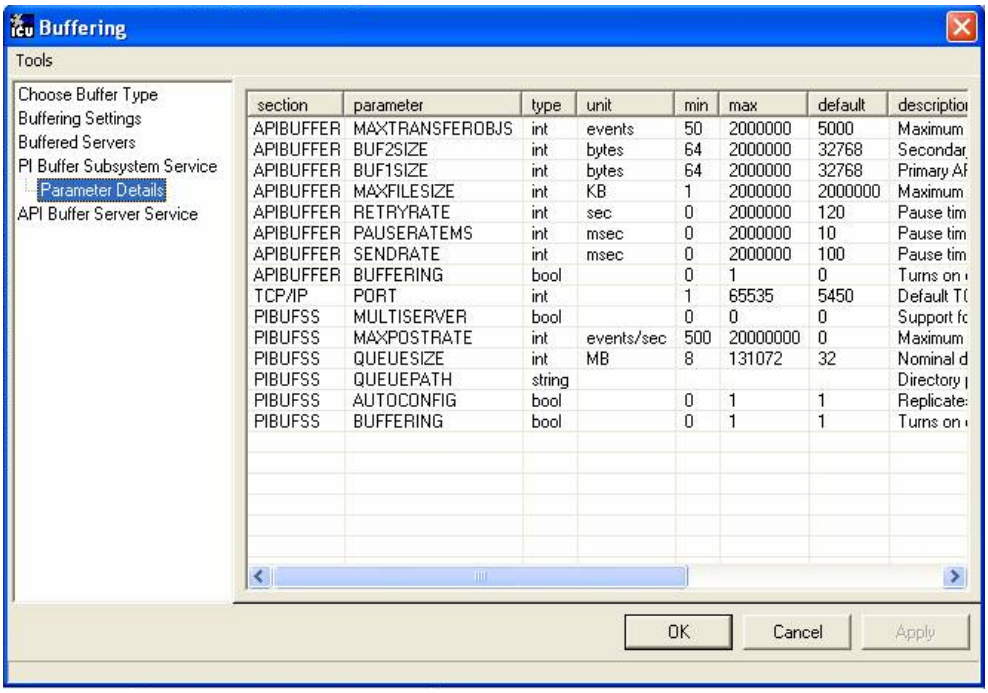
Both the PIBufss and Bufserv applications run as a Service.

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.

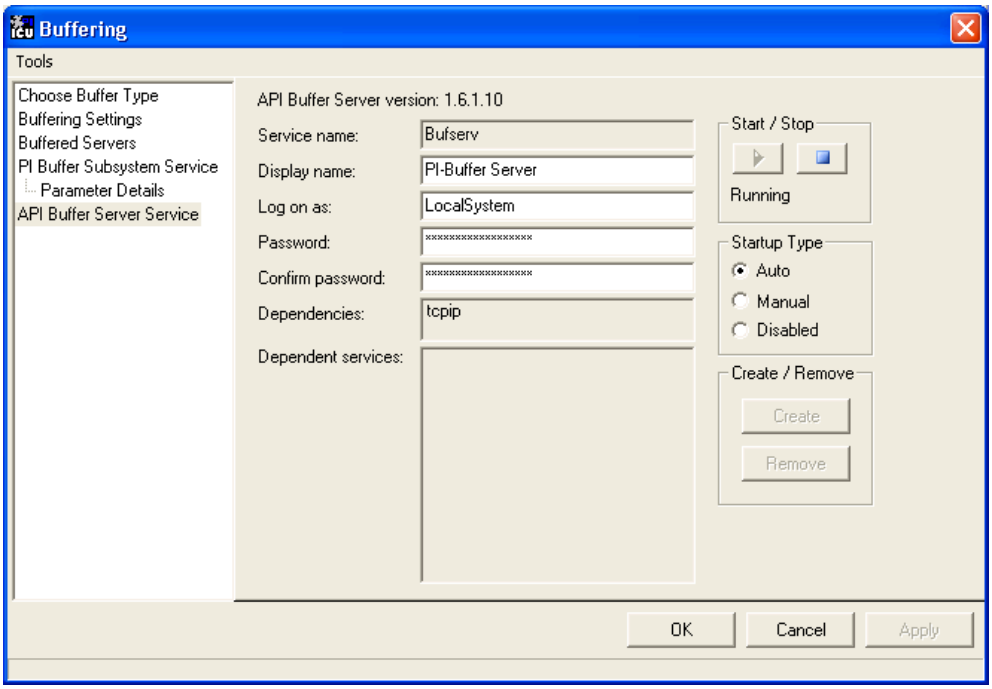




API Buffer Server Service

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.

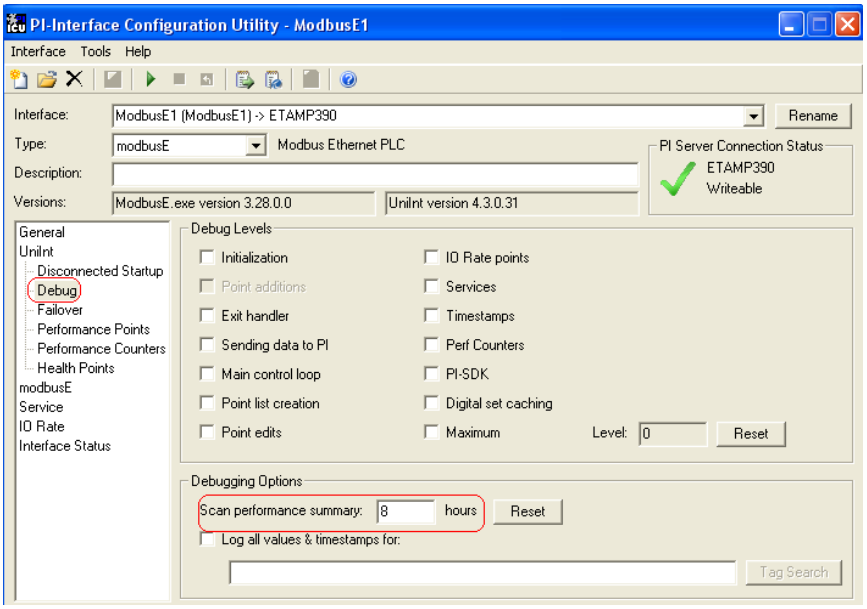

Buffering
Tools
Choose Buffer Type
Buffering Settings
Buffered Servers
PI Buffer Subsystem Service
Parameter Details
API Buffer Server Service
API Buffer Server version: 1.6.1.10
Service name:
Bufserv
Display name:
PI-Buffer Server
Log on as:
LocalSystem
Password:
Confirm password:
Dependencies:
tcpip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

Chapter 14. Interface Diagnostics Configuration

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

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

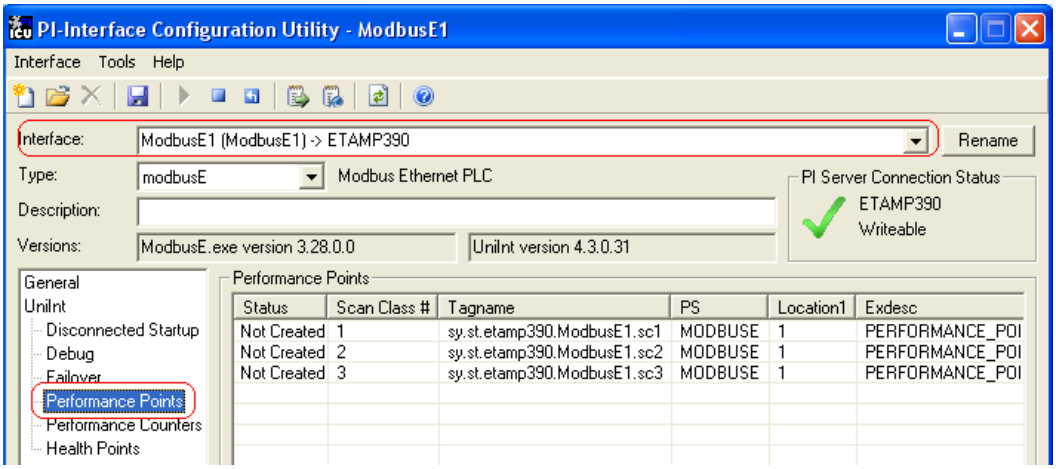
Some of the points that follow refer to a "performance summary interval". This interval is 8 hours by default. You can change this parameter via the *Scan performance summary* box in the *UniInt – Debug* parameter category page:



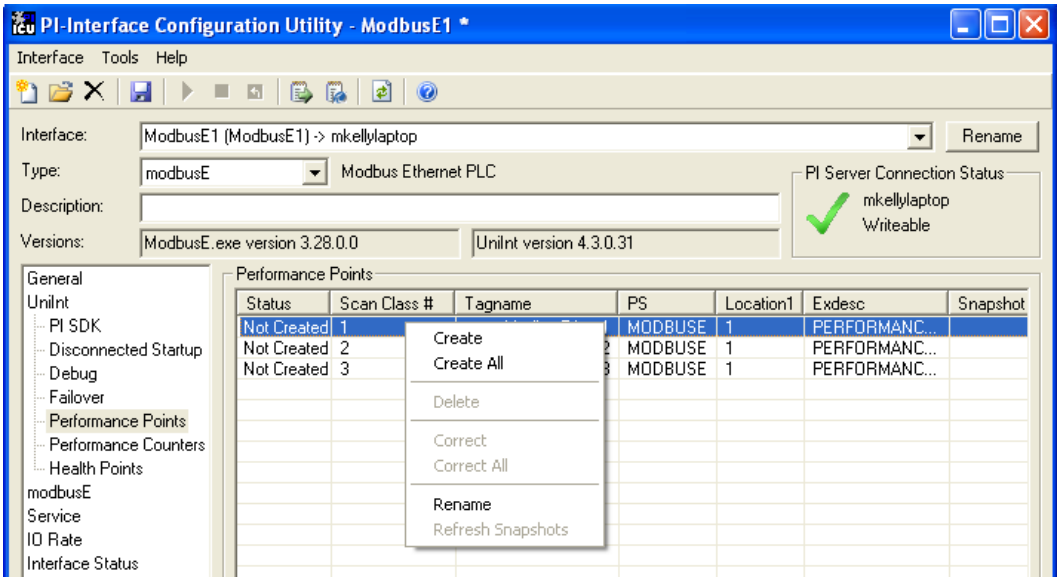
Scan Class Performance Points

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

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



Right-click the row for a particular *Scan Class #* to open the shortcut 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 *Correct All*.

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 the *Scan Class #* column.

Created – Indicates that the Performance Point does exist

Not Created – Indicates that the Performance Point does not exist

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

Scan Class

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

Tagname

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

PS

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

Location1

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

ExDesc

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

Snapshot

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

Performance Counters Points

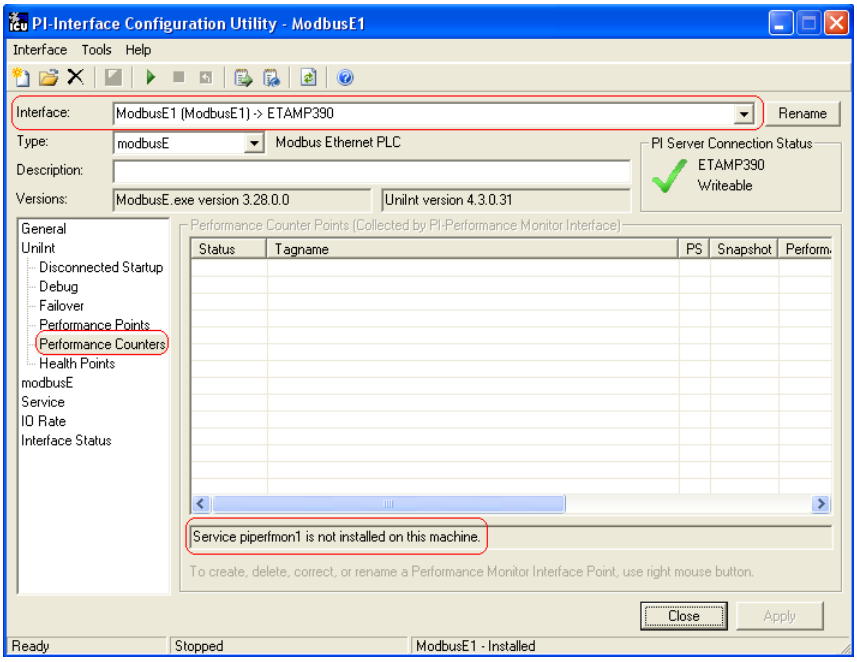
When running as a Service or interactively, this interface exposes performance data via Windows Performance Counters. Such data include items like:

- 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

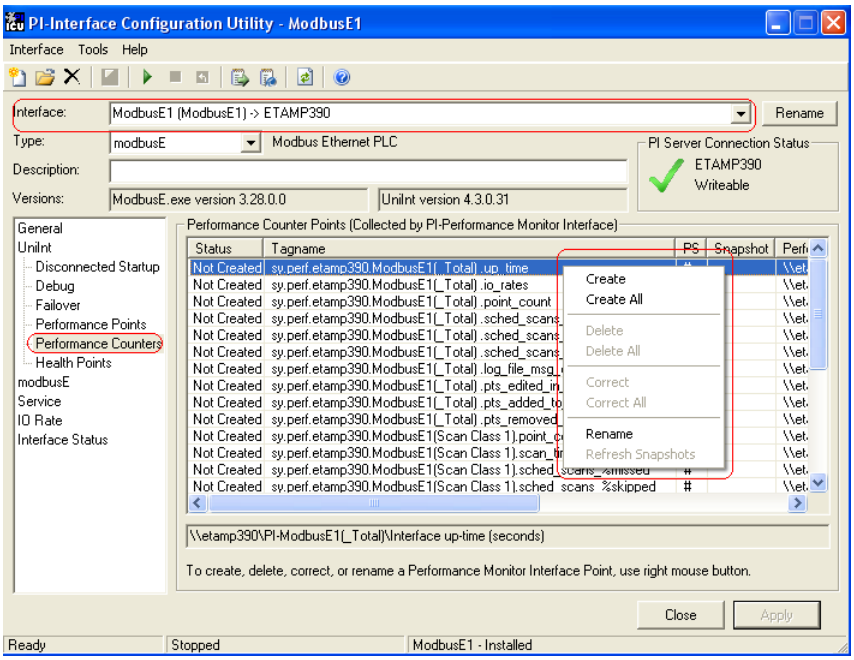
There are two types or instances of Performance Counters that can be collected and stored in PI Points. The first is (_Total) which is a total for the Performance Counter since the interface instance was started. The other is for individual scan classes (Scan Class *x*) where *x* is a particular scan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

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 an "(_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 an "(_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 an "(_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 an "(_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 Device Expected Connections counter.

"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 (that is, 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 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 Server. If the interface is properly communicating with the PI Server then the value of the counter is '0'. If the communication to the PI Server goes down for any reason then the value of the counter will be '1'. Once the interface is properly communicating with the PI Server again then the value will change back to '0'.

"Points added to the interface" (.pts_added_to_interface)

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

"Points edited in the interface"(.pts_edited_in_interface)

The *.pts_edited_in_interface* Performance Counters Point indicates the number of point edits the interface has detected. The interface detects edits for those points whose PointSource attribute matches the **`/ps=`** parameter and whose Location1 attribute matches the **`/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 point attributes is updated and the point is no longer a part of the interface configuration. For example, changing the PointSource, Location1, or Scan attribute can cause the tag to no longer be a part of the interface configuration.

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

The *.Points_Stale_10min* Performance Counters Point indicates the number of good points that have not received a new value in the last 10 minutes. If a point is Good, then it will remain in the good list until the Stale timeout elapses. At this time if the point has not received a new value within the Stale Period then the point will move from the Good count to

the Stale count. Only points that are Good can become Stale. If the point is in the In Error count then it will remain in the In Error count until the error clears. As stated above, the total count of Points Good, Points In Error, and Points Stale will match the Point Count for the interface.

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

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

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

The *.Points_Stale_60min* Performance Counters Point indicates the number of points that have not received a new value in the last 60 minutes. For a point to be in the Stale 60 minute count it must also be a part of the Stale 10 minute and 30 minute count.

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

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

Performance Counters for (Scan Class x) only

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

A *.Device_Scan_Time* Performance Counter Point is available for each scan class of this interface.

The *.Device_Scan_Time* Performance Counters Point indicates the number of milliseconds the interface takes to read the data from the foreign device and package the data to send to 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)

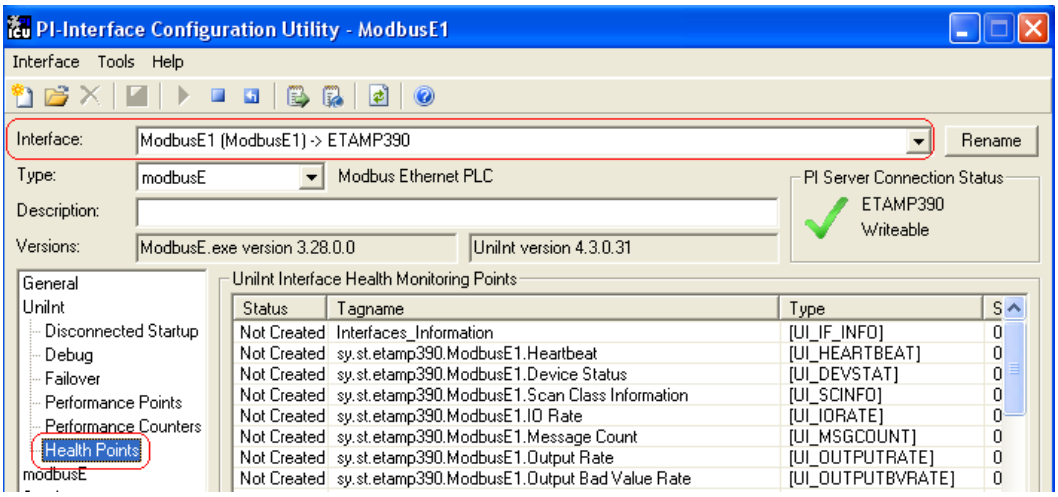
A *.scan_time* Performance Counter Point is available for each scan class of this interface.

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.

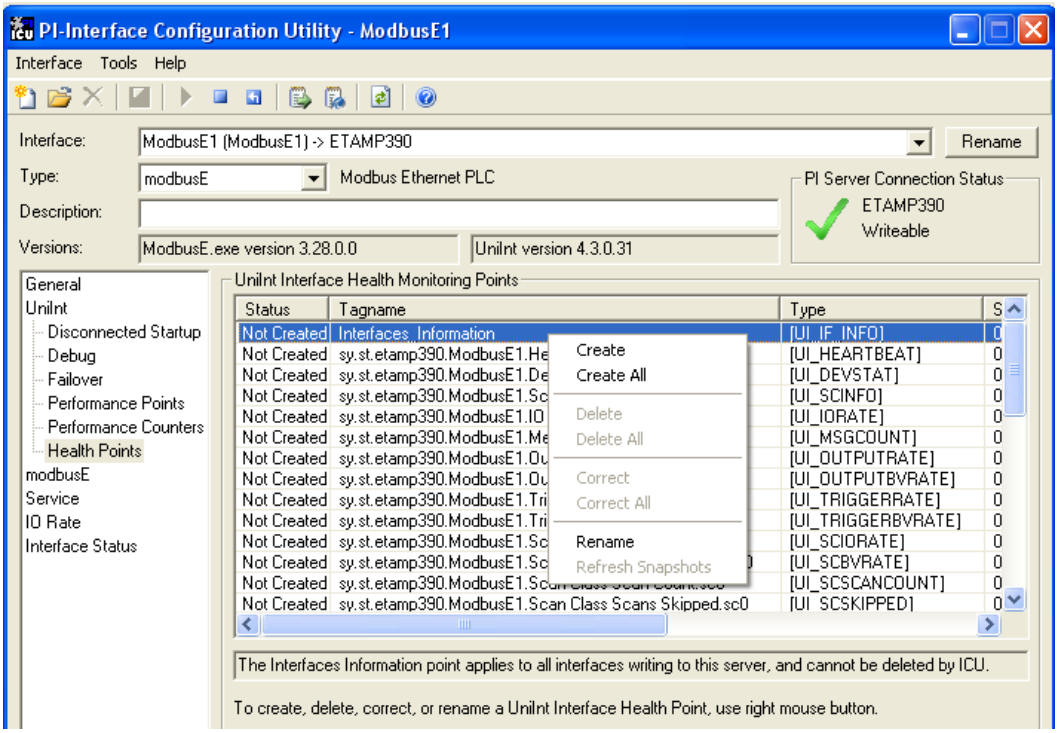
The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).scan_time"`) refers to scan class 1, "(Scan Class 2)" refers to scan class 2, and so on.

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 shortcut 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 [UI_DEVSTAT] Health Point provides an indication of the connection status between the interface and the PLC(s) or PLC gateway. The possible values for this string point are:

- "`1 | Starting`" – The interface remains in this state until it has successfully collected data from its first scan.
- "`Good`" – This value indicates that the interface is able to connect to all of the devices referenced in the interface's point configuration. A value of "Good" does not mean that all tags are receiving good values, but it is a good indication that there are no hardware or network problems.
- "`4 | Intf Shutdown`" – The interface has shut down.

The interface updates this point whenever the connection status between the interface and the PLC(s) or PLC gateway changes.

[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 log file since start-up. In general, a large number for this point indicates that the interface is encountering problems. You should investigate the cause of these problems by looking in log messages.

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

[UI_POINTCOUNT]

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

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

[UI_OUTPUTRATE]

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

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

[UI_OUTPUTBVRATE]

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

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

[UI_TRIGGERRATE]

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

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

[UI_TRIGGERBVRATE]

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

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

[UI_SCIORATE]

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

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

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

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

[UI_SCBVRATE]

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

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

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

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

[UI_SCSCANCOUNT]

You can create a [UI_SCSCANCOUNT] Health Point for each scan class in this interface. The ICU uses a tag naming convention such that the suffix ".sc1" (for example,

`sy.st.etamp390.E1.Scan Class Scan Count.sc1`) refers to scan class 1, ".sc2" refers to scan class 2, and so on.

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

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

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

[UI_SCSKIPPED]

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

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

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

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

[UI_SCPOINTCOUNT]

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

This Health Point monitors the number of tags in a scan class.

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

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

[UI_SCINSCANTIME]

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

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

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

[UI_SCINDEVSCANTIME]

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

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

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

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

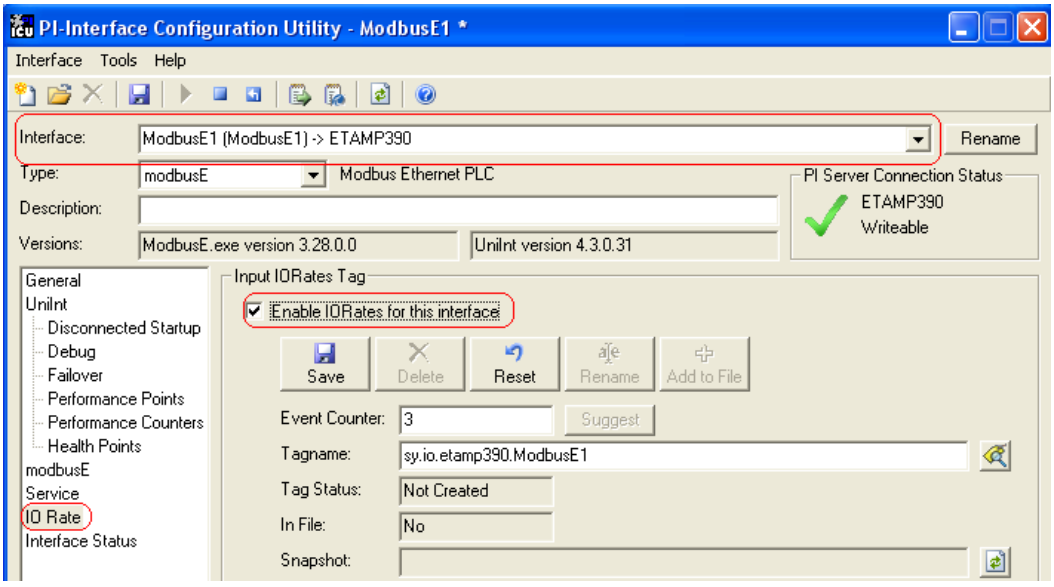
The interface updates the value of this point at the completion of the associated scan.

I/O Rate Point

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

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

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



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.

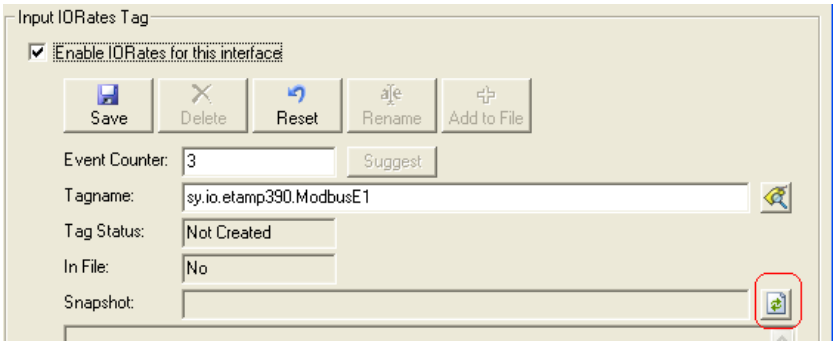
You need to restart the interface in order for it to write a value to the newly created I/O Rate point. Restart the interface by clicking the *Restart* button:

(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 in the *Tagname* box is the name of the I/O Rate tag.

Tag Status

The *Tag Status* box 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 Server

In File

The *In File* box 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* box is updated when the *IORate* page is selected, and when the interface is first loaded.

Create/Save

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

Delete

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

Rename

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

Add to File

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

Search

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

Interface Status Point

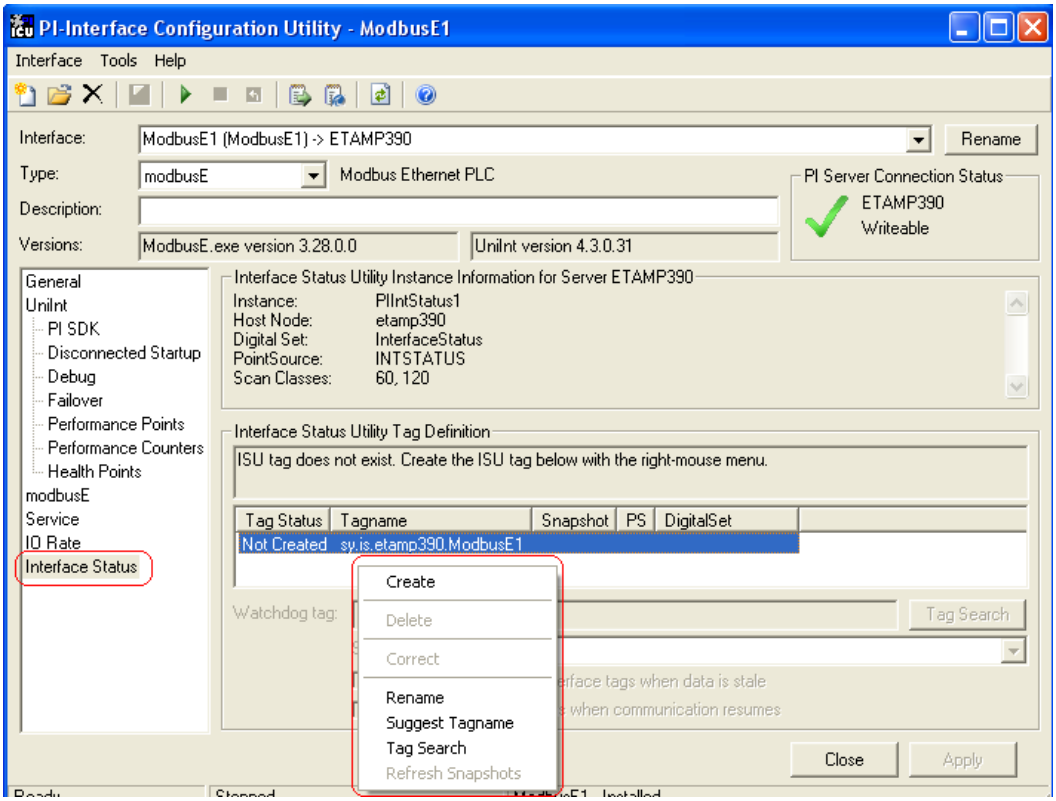
The PI Interface Status Utility (ISU) alerts you when an interface is not currently writing data to the PI Server. This situation commonly occurs if

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

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

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

If you have used the ICU to configure the PI Interface Status Utility Interface on the PI Server Node, the ICU allows you to create the appropriate ISU point. Select this interface from the *Interface* drop-down list and click *Interface Status* in the parameter category pane. Right-click on the ISU tag definition window to open the shortcut 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 open the shortcut menu and select *Correct*.

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

Interface Informational Messages

Message	INFO> /ExcMax=n
Meaning	This message displays the value of the Exception maximum time that will be passed to the SBP.

Message	INFO> Will try to resubscribe every n seconds after the first resubscribe failure
Meaning	This message displays the resubscribe rate which will be used by the interface.

Message	INFO> SBP version detected is n.n.n.n
Meaning	This message is logged at startup to indicate the version of the SBP software.

Interface Warning Messages

Message	`WARNING> Error getting SBP version from the registry`
Meaning	This warning occurs if an error occurs while attempting to read the SBP version from the registry. This could indicate that the SBP software is not installed correctly. The interface will attempt to continue to run.

Interface Error Messages

Message	`ERROR > Unable to initialize secondary log file`
Meaning	The interface was unable to create and initialize the secondary log file. Verify that the file specified is correct and that it can be written.

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 and UNIX, descriptions of system and PI errors can be obtained with the pidiag utility:

Windows: `\PI\adm\pidiag /e error_number`

UNIX: `/PI/adm/pidiag -e error_number`

UniInt Failover Specific Error Messages

Informational

Message	`16-May-06 10:38:00` `PiMax 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 Server while in this state. The amount of data queued while in this state is determined by the failover update interval. In any case, there will be typically no more than two update intervals of data in the queue at any given time. Some transition chains may cause the queue to hold up to five failover update intervals worth of data.

Message	`16-May-06 10:38:05` `PiMax 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 Server as it is received. This message also states that there is not a backup interface participating in failover.

Message	16-May-06 16:37:21 PiMax 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 Server as it is received. This message also states that the other copy of the interface appears to be ready to take over the role of primary.

Errors (Phase 1 & 2)

Message	16-May-06 17:29:06 PiMax 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 PiMax 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 PiMax > 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 PiMax 1> The Failover ID parameter (**/UFO_ID**) was found but the ID for the redundant copy was not found
Cause	The **/UFO_OtherID** parameter is not defined or has not been assigned a positive integer value.
Resolution	Change and verify the **/UFO_OtherID** parameter to a positive integer and restart the interface.

Errors (Phase 2)

Unable to open synchronization file

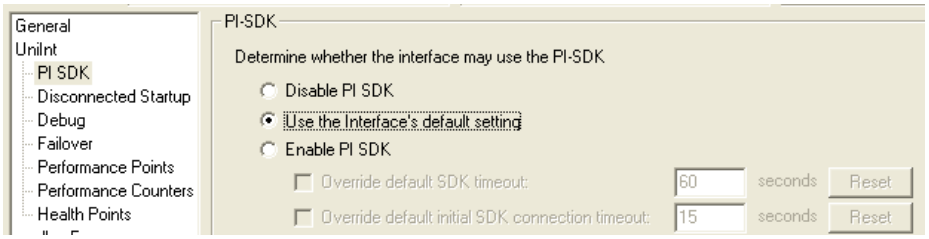
Message	27-Jun-08 17:27:17 PI Eight Track 1 1> Error 5: Unable to create file '\\georgiaking\GeorgiaKingStorage\UnIntFailover\\PIEightTrack_eight_1.dat' Verify that interface has read/write/create access on file server machine. Initializing UniInt library failed Stopping Interface
Cause	This message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

If a remote maxDNA node becomes inoperable, the tags associated with that node will return an error code to PI (I/O Timeout), and will not report any future values to PI until the remote node becomes operable again. Once the remote node is operable, the interface automatically starts sending data to PI.

Prior to version 1.3, there was an issue with the communication error recovery. The situation occurred when there was no active DPU. There are several situations where there is no active DPU.

One situation is when there is only one DPU, and that DPU either fails or is pulled out for replacement or maintenance. Another situation is when there is more than one DPU, but backup mode is not enabled. In this case, there may be other DPUs on the network, but they will not become active because backup mode is not enabled.

Prior to version 1.3, when the DPU once again became available, the interface would not resume collecting data. This was a subscription issue, and has been corrected in version 1.3.

Appendix D. Troubleshooting

If the interface is behaving in an unexpected manner, check the `pipc.log` file and the user-specified log file. (Not all error messages are written to the screen). In general, the user-specified log file will contain greater detail then the `pipc.log` file.

Frequently Asked Questions

Q. As soon as I start the interface, it exits. What could cause this?

A. First, check the log files. A detailed explanation may be found there. Common causes may be:

- PI is not running on the specified host
- maxDNA is not running locally
- a required interface DLL is missing (see Interface Installation)

Q. Why does it take several minutes to see new values after I add a new tag while the interface is running?

A. The interface checks for changes in the PI point database every two minutes. Any new values will not be sent to PI until the first scan after the tag was added.

Q. Why do I constantly get a timed-out status for tags on a remote node?

A. Most likely the remote maxDNA node is not running. If the node is running, check the tag configuration to make sure that the proper node, tag, and field have been specified.

Q. My PI values are not being updated, but the log files show that data is being sent to the interface.

A. Check the permissions of your PI tags. If the user of the interface does not have access to read and write the given PI tags, they will not be updated.

Message Logging

The maxDNA interface provides extensive run-time operation logging facilities. The logging facility provides the following capabilities:

- Multiple detail levels: low, medium, high, none, all;
- A forced writes option that, when enabled, commits all data to the drive after each write (slows the process significantly);
- A screen logging option that will echo the log entries to the console;
- Ability to append or overwrite existing log;
- Circular log file format with selectable size; and
- A common log file viewer for Windows systems.

Run Time Logging Configuration

You can change the operation of the logging facilities at runtime by creating several Logging Tags. The logging tags provide a mechanism for changing the logging configuration during run-time. When the logging configuration needs to be changed, a value can be written to the appropriate tag. For each possible logging configuration change, there is a specific PI tag.

The following table describes the configuration changes that are permitted and the associated tags and values:

Configuration Parameter	Tagname	Appropriate Values
Detail level	$LOG_LEVEL	9 = All logs 8 = No logs 3 = High detail 2 = Medium detail 1 = Low detail
Log to screen	$LOG_SCREEN	0 = off, 1 = log to screen
Enable forced writes	$LOG_FORCED	0 = off, 1 = writes committed immediately

All logging tags must be configured as output tags of type Integer. No source point parameter is required.

Appendix E. 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 Servers.

N-Way Buffering

If you have PI Servers 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 Servers in a PI Collective. (Bufserv also supports n-way buffering to multiple PI Servers however it does not guarantee identical archive records since point compressions attributes could be different between PI Servers. 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.

Interface Node

An interface node is a computer on which

- the PI API and/or PI SDK are installed, and
- PI Server programs are not installed.

PI API

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

PI Collective

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

PIHOME

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

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

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

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

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

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

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

PIHOME64

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

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

PI 64-bit interfaces reside in a subdirectory of the `Interfaces` directory under *PIHOME64*.

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

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

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

PI Message Log

The PI message log is the file to which OSIsoft interfaces based on UniInt 4.5.0.x and later write informational, debug and error messages. When a PI interface runs, it writes to the local PI message log. This message file can only be viewed using the PIGetMsg utility. See the *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 Server. Some PI interfaces, in addition to using the PI API, require the use of the PI SDK.

PI Server Node

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

PI SMT

PI SMT refers to PI System Management Tools. PI SMT is the program that you use for configuring PI Servers. A single copy of PI SMT manages multiple PI Servers. PI SMT runs on either a PI Server Node or a 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 Server. For a given timestamp, a PI point holds a single value.

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

Date	Author	Comments
18-May-1999	TC	First draft
12-Aug-1999	TC	Second draft
18-Oct-1999	JFZ	Updated document based on trip to test at MCS
04-Nov-1999	TC	Revision 2 – Draft 1
05-Jul-2000	EW	Added explanation of /excmax flag
26-Apr-2002	CG	Formatting, TOC
15-Nov-2002	LNG	Updated for version 1.3. Added /subchk flag and clarified that MaxStation software does not need to be running on the same machine as the interface.
26-Nov-2002	CG	Fixed headers & footers; fixed page #s; changed some section headings; does not conform to standard document format
01-Aug-2003	LNG	Added “and greater” to the version, for 1.3.0.1 release. Added version requirement for maxAPPS. Added supported Windows versions. Added /pisdk option to the list of startup parameters.
17-May-2004	LNG	Updated version to 1.4.1.0
17-May-2004	CG	1.4.1.0 Rev C: Fixed headers & footers; clarified running as a service; used current skeleton formatting
19-May-2004	CG	1.4.1.0 Rev D: Fixed the part number
27-May-2004	CG	1.4.1.0 Rev E: Change name from the Max1000+Plus Interface to the maxDNA Interface
03-Jan-2008	JH	1.4.2.0 Rev A – Updated to skeleton version 2.5.6 and added failover.
20-Nov-2012	SBranscomb	Version 1.4.2.0 Revision B; Updated Manual to Skeleton Version 3.0.35.
18-Jul-2013	MHruzik	Updated Manual to Skeleton Version 3.0.36
12-Sep-2013	ZRyska	Corrections, Updated table of content and file has been saved as final.