PI Interface for HTML

Version 2.3.0.x

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

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

Published: 04/2014

Table of Contents

Chapter 1. Introduction

The PI Interface for HTML (HyperText Markup Language) allows a user to collect data that is available in HTML-formatted text. This HTML text can be retrieved by the interface via HTTP (HyperText Transfer Protocol), HTTPS (HyperText Transfer Protocol Secure), FTP (File Transfer Protocol), Gopher, or from the interface node's local file system.

The PI Interface for HTML has the capability of storing a script describing how to get to a particular web page. This is useful for pages that require a login, or for pages that are created by filling out a form.

The interface can either provide its own timestamps for data, or it can parse timestamps from the HTML. Timestamps should be in a format that can be understood by the Visual Basic function CDate.

Starting with version 1.1.0, the HTML interface supports user-developed plug-ins for dynamically generating URLs, and for post-processing timestamps and values. Starting with version 1.2.0.4, the HTML interface supports one more plug-in routine for modifying the downloaded HTML before parsing it.

The software requirements for the PI Interface for HTML are Microsoft Internet Explorer 5.5 or later, the PI Interface Configuration Utility (PI ICU), the PI SDK (which installs the PI API library), Visual C++ 10.0 runtime libraries, and the .NET Framework 4.0.

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 Data Archive manuals
- *PI API Installation Instructions* manual
- *UniInt Interface User Manual*
- *Regular Expressions Tutorial*
- *PI Interface Configuration Utility User Manual*

Supported Operating Systems

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

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

Please contact OSIsoft Technical Support for more information.

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

Supported Features

Feature	Support
Interface Part Number	PI-IN-OS-HTML-NTI
Auto Creates PI Points	No
Point Builder Utility	No
ICU Control	Yes
PI Point Types	float16 / float32 / float64 / int16 / int32 / string / digital

Feature	Support
Sub-second Timestamps	No
Sub-second Scan Classes	No
Automatically Incorporates PI Point Attribute Changes	Yes
Exception Reporting	Yes
Outputs from PI	No
Inputs to PI:	Scan-based / Event Tags
Supports Questionable Bit	No
Supports Multi-character PointSource	Yes
Maximum Point Count	Unlimited
* Uses PI SDK	No
PINet String Support	No
* Source of Timestamps	HTML Page / Current Interface Node Time
History Recovery	No
* UniInt-based * Disconnected Startup * SetDeviceStatus	Yes No Yes
* Failover	UniInt Failover (Phase 2) cold
* Vendor Software Required on Interface Node / PINet Node	Yes
Vendor Software Required on Foreign Device	No
Vendor Hardware Required	No
Additional PI Software Included with interface	Yes
Device Point Types	String
Serial-Based interface	No

** See paragraphs below for further explanation.*

Uses PI SDK

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

Source of Timestamps

Many web sites will provide timestamps with any data they have published. Some will not. The user can configure whether to read a timestamp from the web page or whether to just use the time the HTML page was read by the interface.

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.

SetDeviceStatus

For a Health Tag with an Extended Descriptor attribute that contains [UI_DEVSTAT], the interface writes the following values:

- "1 | Could not read web page." – If the interface cannot connect to the web site, this message is written to the Health tag.

Refer to the `UniInt Interface User Manual.pdf` file for more information about how to configure Health Tags.

Failover

- UniInt Failover Support (Phase 2 Cold failover)

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

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

Vendor Software Required

The PI Interface for HTML takes advantage of technology used in Microsoft Internet Explorer. Version 5.5 or later of Internet Explorer is required for the interface to function properly. It is available from Microsoft's web site at http://www.microsoft.com.

Security Note: To take advantage of the latest security features, we recommend using the latest version of Microsoft Internet Explorer.

Also required by the PI Interface for HTML is Microsoft's XML Parser, version 6.0 or later. The interface uses an XML file to store much of the interface configuration information. This is available at Microsoft's web site, and is installed by the interface install kit as well as the PI SDK version 1.3.1 or later.

Since version 2.3.0.0, the PI Interface for HTML requires the .NET Framework 4.0 or later. The .NET Framework 4.5 is included with newer versions of Windows. For older versions of

Windows, .NET Framework 4.0 is available for download either from Microsoft's web site or by using Windows Update.

In most cases, this interface will be used to retrieve data from a web site. In that case, a remote web server is required to serve the data that will be used by the interface.

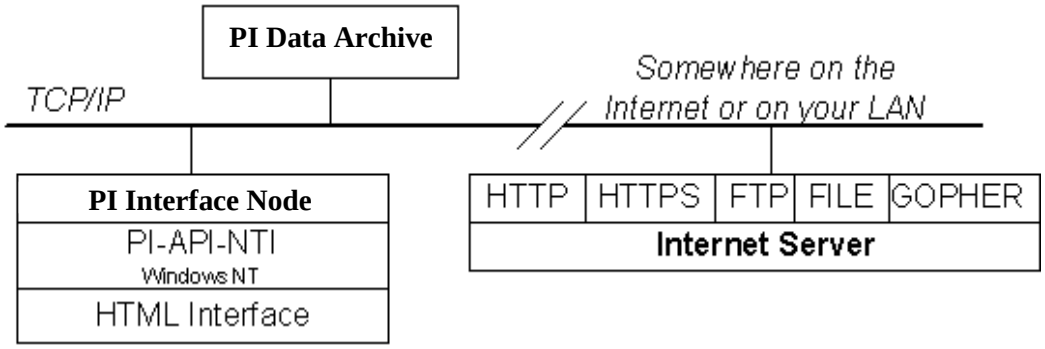
Additional PI Software

The PI Interface Configuration Utility is recommended to configure the PI Interface for HTML. It is included with the interface, and it can be used to configure some other interfaces. As of this time, the PI ICU requires a PI Data Archive of version 3.3 or later. For PI Data Archives earlier than version 3.3, there is another simpler configuration utility also supplied with the interface.

Device Point Types

Although there are many point types that can be read from a web page, in their native form as text on the page, they are text strings. The PI Interface for HTML parses data into the appropriate data types before sending them to the PI Data Archive.

Diagram of Hardware Connection



Chapter 2. Principles of Operation

CURL

Curl is a freely available library for retrieving HTML pages from the internet. Version 2.0 of the PI Interface for HTML uses Curl as its downloading engine. The Curl library is built into the PI Interface for HTML, and therefore does not require a DLL.

Curl can access pages that require HTTP authentication. This is a new feature in version 2.0. Curl can also go through proxy servers for networks that require going through a proxy to get to the internet. This is also a new feature in version 2.0.

MSHTML

The PI Interface for HTML incorporates Microsoft's Internet Explorer (MSIE) components. MSIE is not a monolithic application. It is composed of several components. The component of interest is the MSHTML component. This is responsible for making the network calls to retrieve an HTML page and for parsing the page. The page is parsed into a hierarchy of objects that are then used by MSHTML to efficiently draw the page in the browser window.

Microsoft makes these components available for reuse by developers who are developing web-browsing applications. The PI Interface for HTML uses the hierarchical object model of the HTML page provided by MSHTML to get the data out of the page. Previous to version 2.0, the PI Interface for HTML also used MSHTML to download the pages from the internet. This function is now handled by the Curl library.

Configuration

The PI Interface Configuration Utility (PI ICU) is used to configure the PI Interface for HTML. The configuration for this interface is done graphically. The user browses to the target web page in a custom browser window, selects (using the mouse) where the data is on that page, and saves that information into a configuration file. That configuration file is read by the interface to figure out where specific data is located on the page.

The configuration file is an XML file stored locally. The configuration file can be generated or edited manually, but it should follow the schema provided with the install kit.

Starting with version 2.3.0.0 the interface is using encryption for Proxy and HTTP Security passwords. If you are using an XML configuration file from previous versions (2.0 or 2.2) and it contains Proxy or/and HTTP Security passwords, you need to recreate the HTML Locator Script configuration item and reenter the passwords before you can use PI Interface for HTML. See HTML Locator Script section for technical details about Locator Script configuration item.

Security Note: The communications between the interface and a source web site are visible to a malicious eavesdropper, which can include the user IDs and passwords used to connect to the web sites. If the target website requires a password, https or VPN should be used to protect the password on the wire. Also

restrict access using permissions and enable security auditing for all access to the configuration file.

The PI ICU is a separate product, but it is included with the distribution package of the PI Interface for HTML. The PI ICU can be used to configure most other PI interfaces. Since many of our interfaces are UniInt-based, the PI ICU has a constant set of configuration parameters it can configure. However, since each of those interfaces also has its own interface-specific parameters, there are plug-ins available for the interfaces that are compatible with the PI ICU that allow configuration of those interface-specific parameters.

Refer to the section in this document titled Configuring the Interface with PI ICU, and refer to the PI Interface Configuration Utility User Manual for more information about the PI ICU.

If the target PI Data Archive is not version 3.3 or later, the PI ICU will not work with that PI Data Archive. This is because the PI ICU makes extensive use of the PI Module Database, which was added in PI Data Archive 3.3. In this case, there is also a simpler configuration tool provided specifically for configuring the HTML interface.

Interface Operation

The PI Interface for HTML uses the configuration settings created by the PI ICU to find data on the specified HTML page. There is a series of steps to dig through the HTML to get to the correct location on the page where the data is.

First, the interface downloads the correct page. In the process of configuring the interface using the PI ICU (or with the simpler configuration tool), the ICU will have recorded a series of steps the user took to find the right target HTML page. The PI Interface for HTML follows those steps that were recorded to get to the same page.

Next, the interface uses the MSHTML component to parse the HTML page into an object model. The important thing about this step is that the HTML text is converted into a hierarchical object model. This hierarchical view of the web page may be queried for a particular node. During the configuration process, the PI ICU records the exact node that the user selected. Then, after the MSHTML component has parsed the page and presented the interface with this hierarchical model, the interface navigates to the same node in the object model and extracts whatever data is there.

Timestamps as well as data can be extracted from a web page. It is possible to associate timestamps with data from a web page to form a complete timestamp-value pair.

For those systems that do not support the PI ICU, a simpler configuration tool is available.

Pattern Matching

The PI Interface for HTML uses regular expression (regexp) pattern matching in order to allow you to do some more advanced searching in the HTML page for data. In some cases, to select exactly the correct data, regexp is required. Take the following snippet of HTML code:

```
<TABLE>
    <TR VALIGN="BOTTOM">
          <TD COLSPAN=3>Weather data for December 12, 2001 12:32pm</TD>
    </TR>
    <TR>
          <TH>Temperature</TH>
          <TH>Humidity</TH>
          <TH>Barometric pressure</TH>
    </TR>
    <TR>
          <TD><B>51</B> °F</TD>
          <TD><B>72</B> %</TD>
          <TD><B>29.97</B> inHg</TD>
</TR>
</TABLE>
```

The data of interest is the timestamp, and the three values. The way data is retrieved from MSHTML, the timestamp would actually be returned to the interface as "Weather data for December 12, 2001 12:32 pm". This is because MSHTML uses the HTML tags as delimiters for the text. In this case, there are no HTML tags separating the "Weather data for" and the actual date part. When the interface tries to parse that into a date, as error will occur, because of the leading text. Pattern matching and substitution can be used to search through this text and select only the data you are interested in.

The values would be read fine without having to use pattern matching, because the numbers themselves are stored inside the bold (<B>) tags. So even though there is text right next to the numbers when viewed in the web browser, in the HTML code, the digits are delimited from the units of measure by HTML tags.

This topic is described in more detail in the Regular Expressions Tutorial. There are techniques and examples for many common situations where pattern matching may be required to have the interface correctly gather the data you want to gather.

Plug-ins

Starting with version 1.1.0, the PI Interface for HTML supports user-created plug-ins. See Appendix C Plug-In Architecture for technical details about how to create these plug-ins using COM. There are two uses for plug-ins.

The first use for plug-ins is to dynamically generate URLs during interface operation. Many times, the target web page will not have a constant URL. For example, a page that includes the date will have a different URL every day. One day, the desired web page is `http://www.yoururl.com/pricing_data_04202002.html`. The next day's data, however, might be found at `http://www.yoururl.com/pricing_data_04212002.html`. The PI Interface for HTML will check with the plug-in to determine the correct URL.

The second use for plug-ins is to post-process timestamps and values. There are some timestamps that just cannot be parsed by the interface. Other times, the timestamp may need to be tweaked just a little. Other times, there may be some kind of convention used by a page, where the reader can easily tell what time the page is talking about, but a machine cannot. For example, there may be a page that gives data at 5-minute intervals, and the interval is reported on the page (1-12) instead of the actual timestamp. For values, there may be data that needs to be massaged before it is sent to the PI Data Archive. The interface will report the timestamps and values read from the page to the plug-in, and the plug-in will perform some operation on the data, and return the timestamp and value back to the interface.

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 .NET Framework 4.0 has been installed.
2. Confirm that you can use PI SMT to configure the PI Data Archive. You need not run PI SMT on the same computer on which you run this interface.
3. If you are running the interface on an interface node, edit the PI Data Archive's Trust Table to allow the interface to read attributes and point data. If a buffering application is not running on the interface node, the PI Trust must allow the interface to write data.
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. Install Microsoft Internet Explorer version 5.5 or higher (http://www.microsoft.com/windows/ie/).
6. 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.
7. If you are running the interface on an interface node, check the computer's time zone properties. An improper time zone configuration can cause the PI Data Archive to reject the data that this interface writes.
8. Run the ICU or the simpler HTML Interface Configuration Utility to setup timestamp and data markers and configure a new instance of this interface. Essential startup parameters for this interface are

   Point Source (**`/PS=`***`x`*)
   Interface ID (**`/ID=`***`#`*)
   PI Data Archive (**`/Host=`***`host:port`*)
   Scan Class (**`/F=`***`##:##:##,offset`*)
   HTML Config File (**`/htmlconfigfile=`***`<UNC Path>`*)
9. Test the connection between the interface node and the target web page by opening it in Internet Explorer.
10. If you will use digital points, define the appropriate digital state sets.

11. Build input tags for this interface. Important point attributes and their purposes are:

    Location1 specifies the interface instance ID.
    Location2 specifies digital states.
    Location3 is not used.
    Location4 specifies the scan class.
    Location5 is not used.
    ExDesc is not used.
    InstrumentTag is the data marker (or markers) associated with the PI point. Delimit data markers with a semicolon (;).
    PtSecurity must permit read access for the PI identity, group, or user configured in the PI Trust that is used by the interface.
    DataSecurity must permit read access (buffering enabled) or read/write access (unbuffered) for the PI identity, group, or user configured in the PI Trust that is used by the interface.

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

12. Start the interface interactively and confirm its successful connection to the PI Data Archive without buffering. (The DataSecurity attribute for interface points must permit write access for the interface's PI Trust.)
13. Confirm that the interface collects data successfully.
14. 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.
15. Start the buffering application and the interface. Confirm that the interface works together with the buffering application by physically removing the connection between the interface node and the PI Data Archive Node. (The DataSecurity attribute for interface points must permit write access for the *buffering application's* PI Trust or mapping. The interface's PI Trust does not require DataSecurity write permission.)
16. Configure the interface to run as a Service. Confirm that the interface runs properly as a Service.
17. Restart the interface node and confirm that the interface and the buffering application restart.

Interface Diagnostics

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

Chapter 4. Interface Installation

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

After the interface has been installed and tested, Buffering should be enabled on the interface node. Buffering refers to either PI API Buffer Server (Bufserv) or the PI Buffer Subsystem (PIBufss). For more information about Buffering see the Buffering 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 Data Archive node. In this case, the typical procedure is to install the PI Data Archive as an automatic service and install the interface as an automatic service that depends on the PI Update Manager and PI Network Manager services. This typical scenario assumes that Buffering is not enabled on the PI Data Archive node. Bufserv or PIBufss can be enabled on the PI Data Archive node so that interfaces on the PI Data Archive node do not need to be started and stopped in conjunction with the PI Data Archive, but it is not standard practice to enable buffering on the PI Data Archive node. The PI Buffer Subsystem can also be installed on the PI Data Archive. See the *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 `PIHTML.exe` and that the startup command file is called `PIHTML.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, `PIHTML1.exe` and `PIHTML1.bat` would typically be used for instance 1, `PIHTML2.exe` and `PIHTML2.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.

Additional Required Software

Microsoft Internet Explorer

Microsoft Internet Explorer version 5.5 or later is required for the PI Interface for HTML. Its browsing and parsing functionality is used by the interface. This should be installed before

the interface is configured or started. The software is available on Microsoft's web site at http://www.microsoft.com/windows/ie/default.htm.

Microsoft XML Parser

The Microsoft XML Parser (MSXML) version 6.0 or later is also required for the PI Interface for HTML. The configuration file used to store the location of the target web page, as well as the spots on the page where the data is stored, is an XML document. MSXML 6.0 is available from Microsoft's web site and is installed by the OSIsoft Prerequisite kits.

Microsoft .NET Framework 4.0

The Microsoft .NET Framework version 4.0 is required for the PI Interface for HTML to run. The interface is now a managed application, and uses the .NET Framework. The .NET Framework 4.5 is included with newer versions of Windows. For older versions of Windows, .NET Framework 4.0 is available for download either from Microsoft's web site or by using Windows Update.

Interface Directories

PIHOME Directory Tree

The [`PIHOME`] directory tree is defined by the `PIHOME` entry in the `pipc.ini` configuration file. This `pipc.ini` file is an ASCII text file, which is located in the `%windir%` directory.

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

```
[PIPC]
PIHOME=C:\Program Files\PIPC
```

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

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

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

Security Note: Restrict the Windows accounts that can create or write files in the interface and configuration folder.

Interface Installation Directory

The interface install kit will automatically install the interface to:

`PIHOME\Interfaces\HTML\`

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

Interface Installation Procedure

The PI Interface for HTML 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.

```
HTML_#.#.#.#_.exe
```

PI Trust for Interface Authentication

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

Installing Interface as a Windows Service

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

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

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

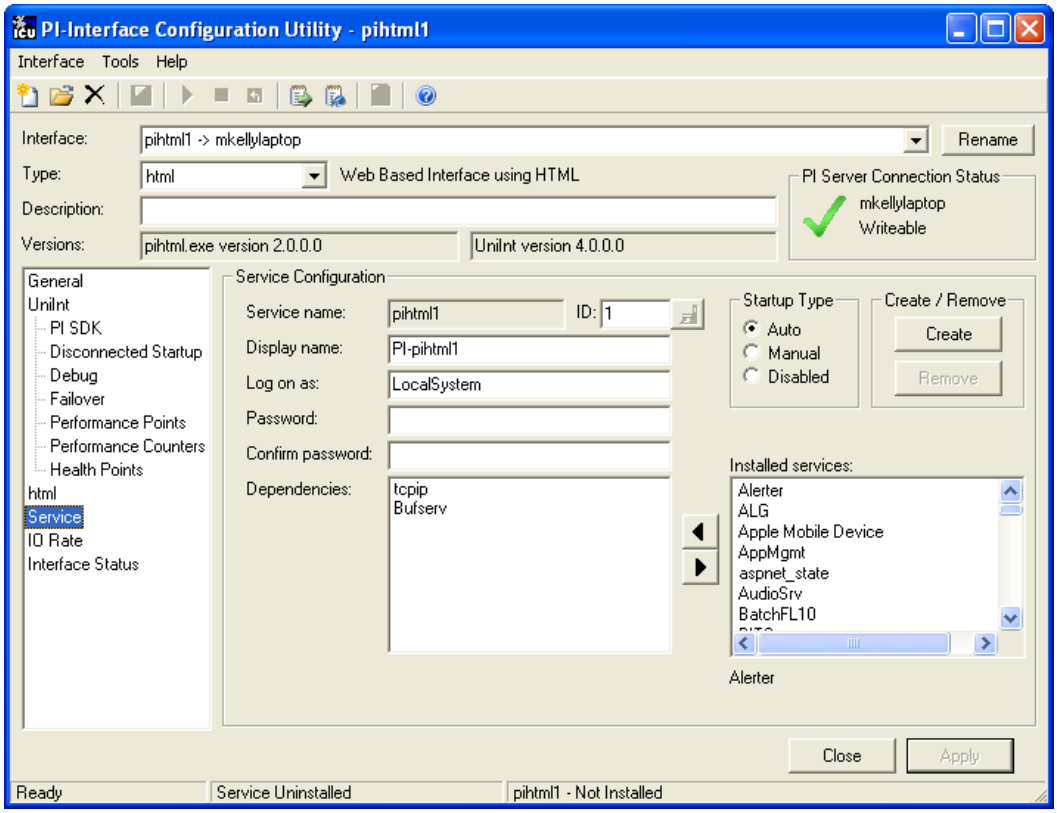
The disadvantage of running the interface service with least privileges is that, depending on the account, the interface service may not be able to create performance counters and extra administrative actions are needed to create and maintain the performance counters. Since performance counters are associated with each scan class, performance counters for the interface instance must be updated after additions or deletions of scan classes by running the interface instance, at least for a short time, from an account that has sufficient privileges to create or delete performance counters.

Log On as Security and DCOM Settings When Running as a Service

Previous versions of the PI Interface for HTML required special security settings to be configured prior to running the interface as a service. Starting with version 2.0, the PI Interface for HTML can run as a service logged on as Local System (the default setting when creating a service), and DCOM does not need to be configured past the default settings.

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.

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

The consequence of increasing security by following this recommendation is that extra administrative actions are needed to create and maintain the performance counters for the interface service. Since performance counters are associated with each scan class, performance counters for the interface instance must be updated after additions or deletions of scan classes by running the interface instance, at least for a short time, from an account that has sufficient privileges to create or delete performance counters.

Unfortunately, the current version of the ICU cannot create a service that runs under the Windows built-in NetworkService account. After ICU creates the interface service, you can change the account with a Windows administrative tool, such as Services on the Control Panel or the sc command-line utility.

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*.

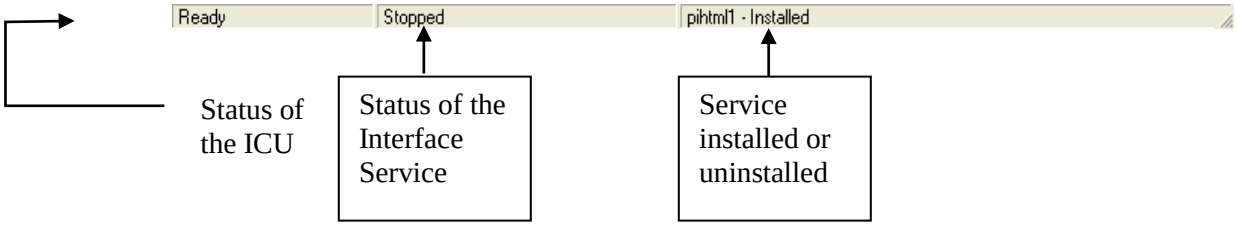
Remove

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:

```
PIHTML.exe /help
```

Open a Windows command prompt window and change to the directory where the `PIHTML.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 Data Archive Node with Bufserv implemented	
Manual service	PIHTML.exe /install /depend "tcpip bufserv"
Automatic service	PIHTML.exe /install /auto /depend "tcpip bufserv"
*Automatic service with service ID	PIHTML.exe /serviceid *X* /install /auto /depend "tcpip bufserv"
Windows Service Installation Commands on an Interface Node or a PI Data Archive Node without Bufserv implemented	
Manual service	PIHTML.exe /install /depend tcpip
Automatic service	PIHTML.exe /install /auto /depend tcpip
*Automatic service with service ID	PIHTML.exe /serviceid *X* /install /auto /depend tcpip

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

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

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

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

The consequence of increasing security by following this recommendation is that extra administrative actions are needed to create and maintain the performance counters for the interface service. Since performance counters are associated with each scan class, performance counters for the interface instance must be updated after additions or deletions of scan classes by running the interface instance, at least for a short time, from an account that has sufficient privileges to create or delete performance counters.

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

Chapter 5. Digital States

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

Digital State Sets

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

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

System Digital State Set

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

Chapter 6. PointSource

The PointSource is a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For example, the string *Boiler1* may be used to identify points that belong to the *MyInt* interface. To implement this, the PointSource attribute would be set to `Boiler1` for every PI point that is configured for the *MyInt* interface. Then, if **`/ps=Boiler1`** is used on the startup command-line of the *MyInt* interface, the interface will search the PI Point Database upon startup for every PI point that is configured with a PointSource of `Boiler1`. Before an interface loads a point, the interface usually performs further checks by examining additional PI point attributes to determine whether a particular point is valid for the interface. For additional information, see the **`/ps`** parameter. If the PI API version being used is earlier than 1.6.x or the PI Data Archive version is earlier than 3.4.370.x, the PointSource is limited to a single character unless the SDK is being used.

Case-sensitivity for PointSource Attribute

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

Reserved Point Sources

Several subsystems and applications that ship with the PI System are associated with default PointSource characters. The Totalizer Subsystem uses the PointSource character `T`, the Alarm Subsystem uses @ for Alarm 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 Data Archive. A single point is configured for each measurement value that needs to be archived.

Point Attributes

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

This document does not discuss the attributes that configure UniInt or PI Data Archive processing for a PI point. Specifically, UniInt provides exception reporting and the PI Data Archive 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 Data Archive 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 Data Archive.
- The first character must be alphanumeric, the underscore (_), or the percent sign (%).
- Control characters such as linefeeds or tabs are illegal.
- The following characters also are illegal: * ’ ? ; { } [ ] | \ ` ' "

Length

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

PI API	PI Data Archive	Maximum Length
1.6.0.2 or later	3.4.370.x or later	1023
1.6.0.2 or later	Earlier than 3.4.370.x	255
Earlier than 1.6.0.2	3.4.370.x or later	255
Earlier than 1.6.0.2	Earlier than 3.4.370.x	255

If the PI Data Archive version is earlier than 3.4.370.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum tag length of 1023, you need to enable the PI SDK. See Appendix B for information.

PointSource

The PointSource attribute contains a unique, single or multi-character string that is used to identify the PI point as a point that belongs to a particular interface. For additional information, see the **/ps** command-line parameter and the PointSource 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, int16, int32, digital, and string point types are supported. For more information on the individual point types, see PI Data Archive Manuals.

Location1

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

Location2

Location2 is used by digital points. Set Location2 = 0 when the text on the page corresponds to the string representation of a digital state. Set Location2 = 1 when the text on the page corresponds to the zero-based integer offset of a digital state in the point's digital state set.

Location3

Location3 is not used by this interface.

Location4

Scan-based Inputs

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

To use event-based scanning, set Location4 to 0 and see the section describing the extended descriptor (ExDesc), below.

Trigger-based Inputs, Unsolicited Inputs, and Output Points

Location4 should be set to zero for these points.

Location5

Location5 is not used by this interface.

InstrumentTag

This field should contain the data marker from which this point will be reading data. This field is not case-sensitive.

If this PI point will be receiving data from multiple data markers, list them all here, delimited by semicolons (;). This is useful when a single point needs to receive multiple values from a page. For example, hourly weather information could be listed as 24 different timestamp/value pairs on the same page, but all values need to go to the same point.

Note: When using multiple markers for a single point, digital state errors are suppressed for that point.

Length

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

PI API	PI Data Archive	Maximum Length
1.6.0.2 or later	3.4.370.x or later	1023
1.6.0.2 or later	Earlier than 3.4.370.x	32
Earlier than 1.6.0.2	3.4.370.x or later	32
Earlier than 1.6.0.2	Earlier than 3.4.370.x	32

If the PI Data Archive version is earlier than 3.4.370.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum InstrumentTag length of 1023, you need to enable the PI SDK. See Appendix B for information.

ExDesc

ExDesc is not used by the PI Interface for HTML for any interface-specific features, but it does enable some functionality present in UniInt interfaces.

Length

Depending on the version of the PI API and the PI Data Archive, this interface supports an ExDesc attribute whose length is at most 80 or 1023 characters. The following table indicates the maximum length of this attribute for all the different combinations of PI API and PI Data Archive versions.

PI API	PI Data Archive	Maximum Length
1.6.0.2 or later	3.4.370.x or later	1023
1.6.0.2 or later	Earlier than 3.4.370.x	80
Earlier than 1.6.0.2	3.4.370.x or later	80
Earlier than 1.6.0.2	Earlier than 3.4.370.x	80

If the PI Data Archive version is earlier than 3.4.370.x or the PI API version is earlier than 1.6.0.2, and you want to use a maximum ExDesc length of 1023, you need to enable the PI SDK. See Appendix B for information.

Performance Points

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

Trigger-based Inputs

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

```
keyword=trigger_tag_name
```

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

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

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

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

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

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

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

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

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

Scan

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

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

Shutdown

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

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

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

Bufserv and PIBufss

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

DataSecurity

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

PointSecurity

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

Output Points

The PI Interface for HTML does not support output points.

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.

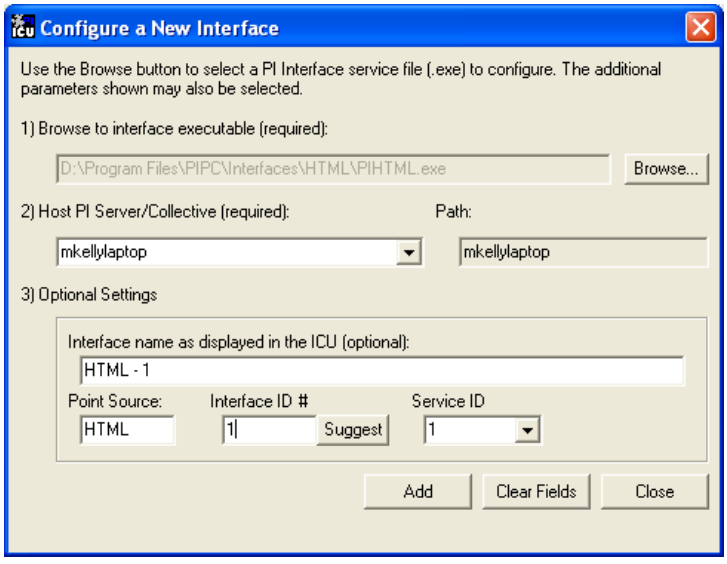
The PI Interface Configuration Utility (PI ICU) provides a tool for configuring the interface startup command file.

Configuring the Interface with PI ICU

Note: PI ICU requires PI 3.3 or later.

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 (`PIHTML.bat`) will be maintained by the PI ICU and all configuration changes will be kept in that file and the PI Module Database. The procedure below describes the necessary steps for using PI ICU to configure the PI Interface for HTML.

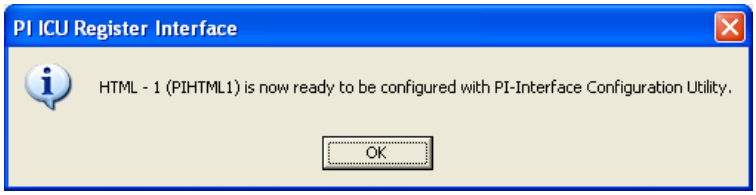
From the PI ICU menu, select *Interface*, then *NewWindows Interface Instance from EXE...*, and then *Browse* to the `PIHTML.exe` executable file. Then, enter values for *Host PI System*, *Point Source*, and *Interface ID#*. A window such as the following results:



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*.

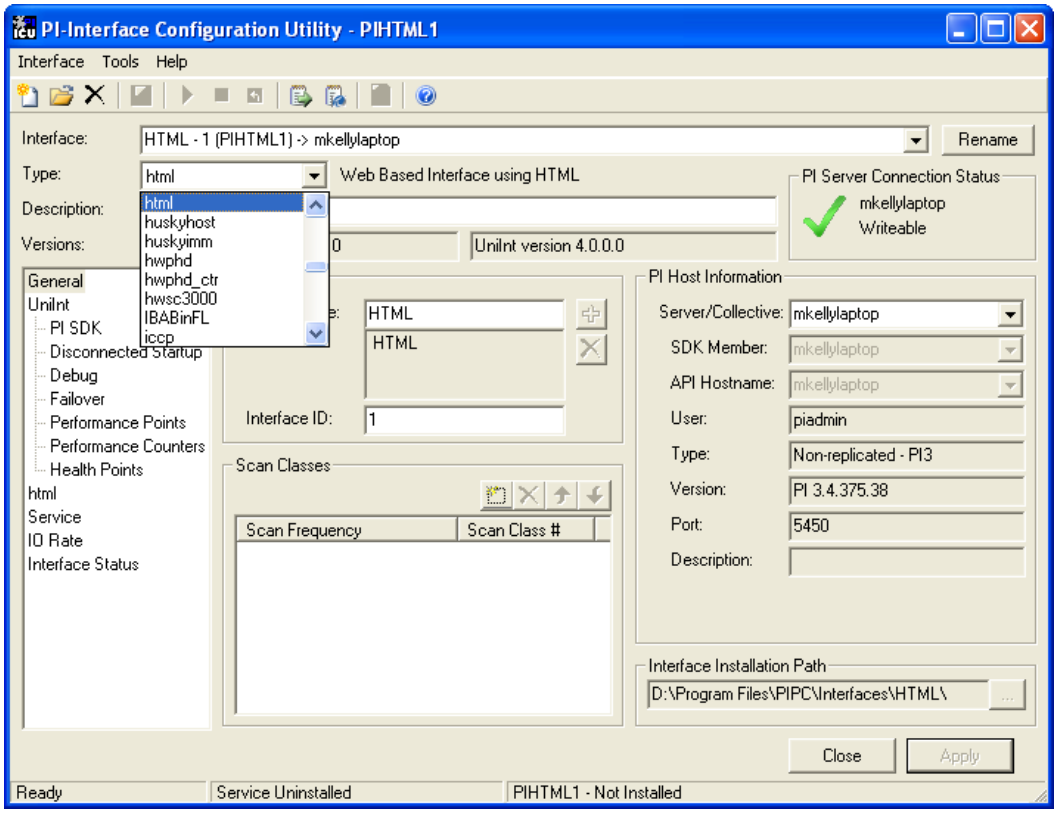
The following message should appear:



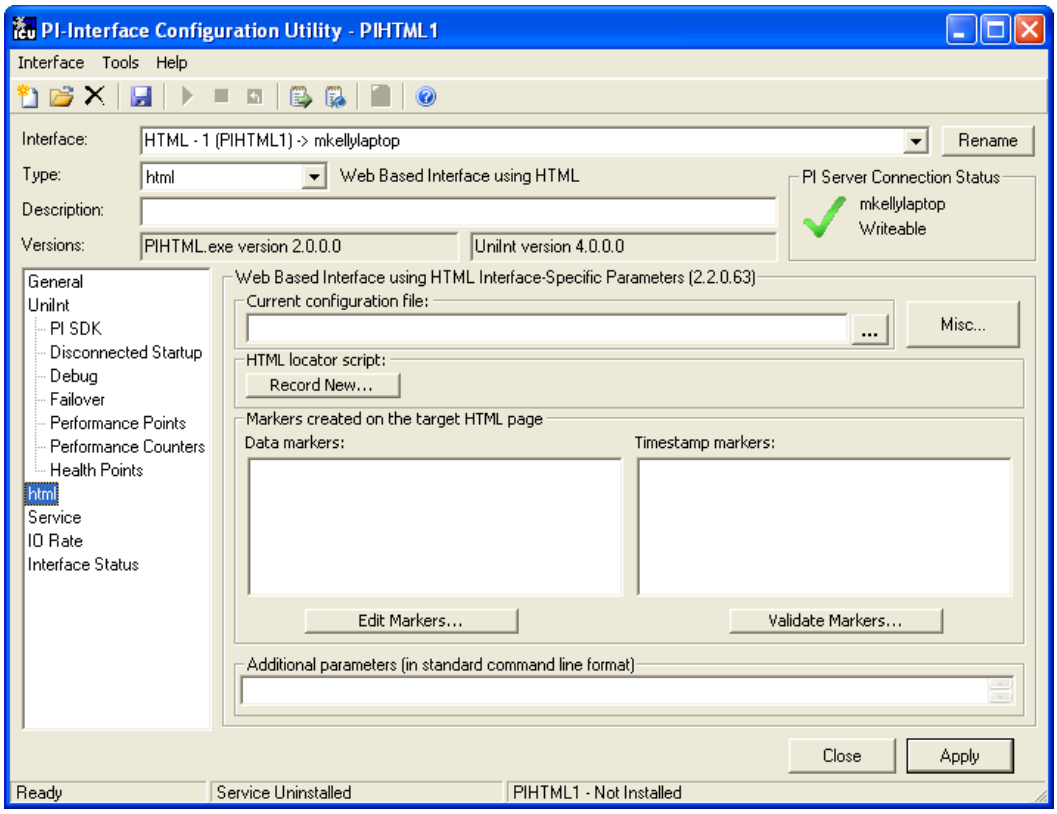
Note that in this example the Host PI Data Archive is mkellylaptop. To configure the interface to communicate with a remote PI Data Archive, select *Connections…* from the PI ICU *Interface* menu and select the default PI Data Archive. If the remote node is not present in the list of PI Data Archives, 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 `html`. If not, use the drop-down box to change the interface *Type* to be `html`.

Click on *Apply* to enable the PI ICU to manage this instance of the PI Interface for HTML.



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



Since the PI Interface for HTML 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.

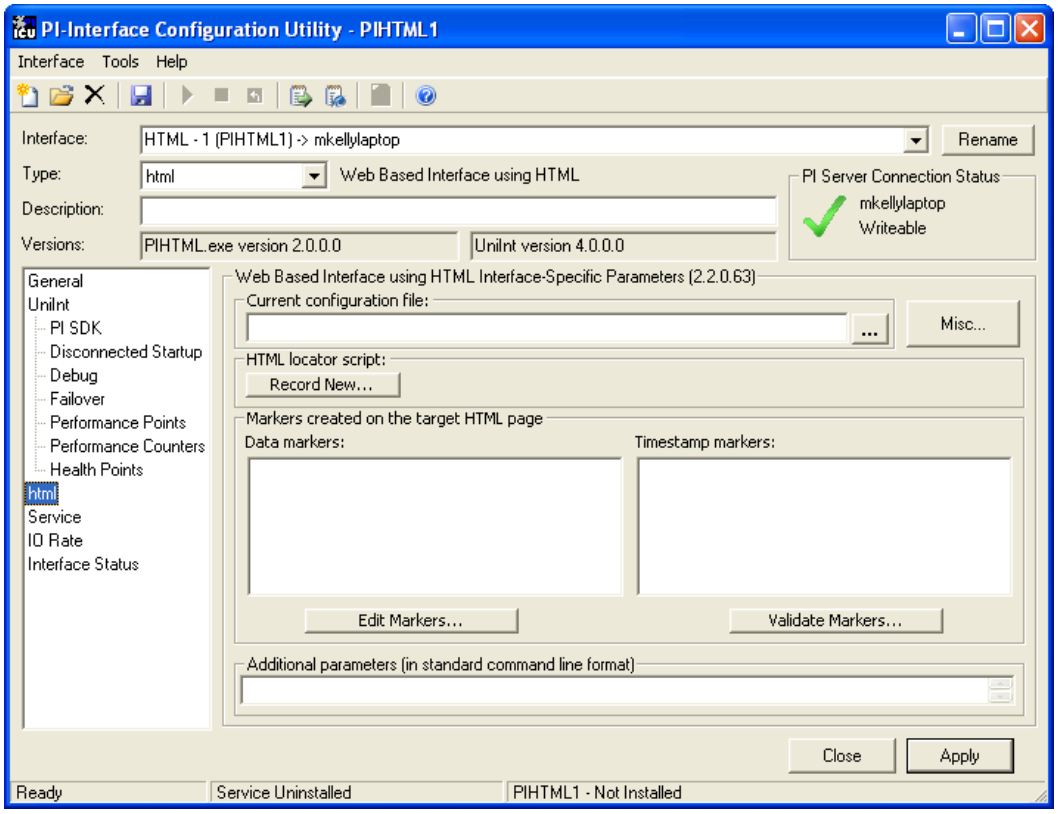
To set up the interface as a Windows Service, use the *Service* page. This page allows configuration of the interface to run as a service as well as to starting and stopping of the interface service. The interface can also be run interactively from the PI ICU. To do that, select *Start Interactive* on 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 *html* page. Once selections have been made on the PI ICU window, press the *Apply* button in order for PI ICU to make these changes to the interface's startup file.

html Interface Page

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

html Interface Page



The PI Interface for HTML - ICU Control has one section. A yellow text box indicates that an invalid value has been entered or that a required value has not been entered.

Current Configuration File

This file is an XML (eXtensible Markup Language)-formatted file that contains detailed interface configuration information. This file should not be edited manually, unless you REALLY know what you are doing. Otherwise, this file is automatically maintained by the PI ICU.

HTML Locator Script

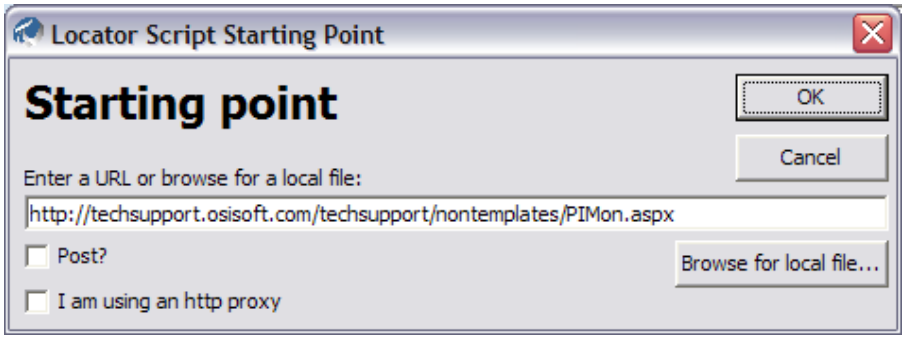
One item stored in the configuration file is the series of steps required to get to the HTML page that the interface will be parsing. This is necessary because there are pages that are not accessible by directly entering a URL. For example, many pages require a login before they will allow a browser to access certain protected information. This functionality will allow a user to graphically walk through what steps are necessary to navigate to a particular page.

Click **Record New** to open a dialog box that prompts you for an initial URL (and possibly proxy information) and then a web browser appears. Navigate the web using the mini-browser window that is provided, clicking on links or filling in forms, until the desired page has been located.

Only one target HTML page can be specified for each instance of the PI Interface for HTML. If data is desired from more than one HTML page, another instance of the interface must be created. This can be done on the first tab of the PI ICU.

Steps for Creating a New HTML Locator Script

1. Make sure the desired configuration file is selected in the text box under the **Current Configuration File** field. This is the file in which the locator script will be held. The ellipsis button will allow you to browse for a file. If a non-existing file is selected, a dialog box will prompt you to create a new configuration file.
2. Click **Record New**. This opens a dialog box asking where the starting point for the web browser is. If you are going through an HTTP proxy server, check the **I am using an http proxy** check box. Enter your proxy server, username, and password. If you need the request to be a POST type request, click the checkbox next to **Post**.

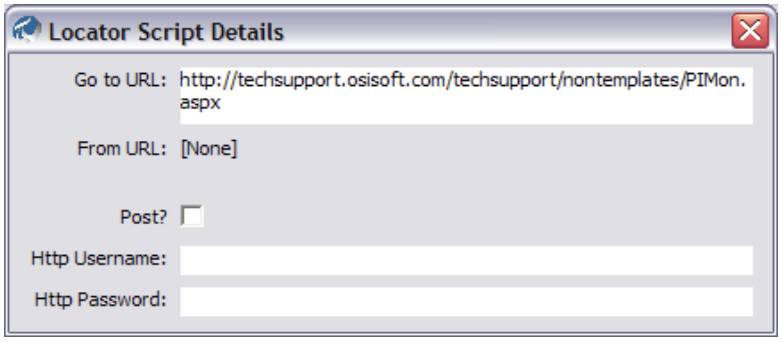


3. Enter the URL to the starting web page, where the navigation to find the target HTML page will begin. Click **OK**.

Note: If you select a file on the local file system, the format of the URL must be file:`///C:/Path/To/My/File.html`.

---

4. Use the web browser window that opens to navigate to the target page. Your actions will be recorded in a list at the bottom of the page.
5. To modify the attributes of the pages navigated to (like the URL, whether the request will be a GET request or a POST request, and the http authentication username and password for getting to a particular page), click the URL of the entry you want to modify in the list box in the upper-right corner of the navigation window and modify those settings.



6. When you are finished, click **Finished**. To revert to the previous locator script, press **Cancel**.

Markers Created on the Target HTML Page

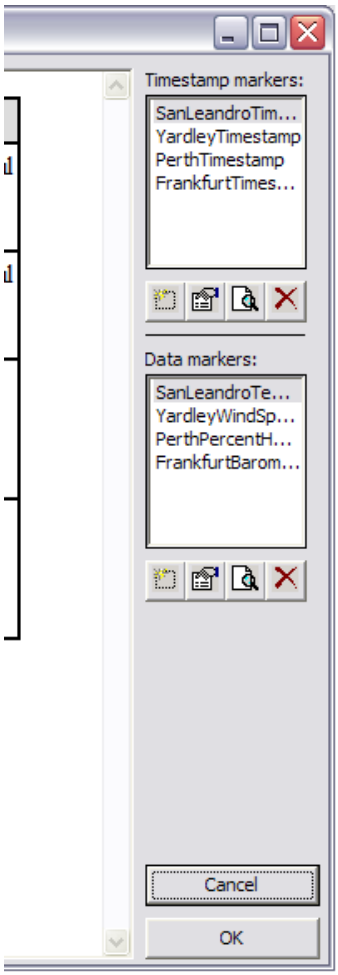
Once an HTML page has been retrieved, the next step is to determine where on that page the data is located. The idea is that you highlight certain places on the rendered HTML page, and the locations of those selections are remembered by the configuration utility and saved in the configuration file. These locations on the page are called markers. By clicking **Edit Markers**, the PI ICU displays the HTML page in a separate window. Select where one piece of data (timestamp data or value data) on that page is located. After selecting a piece of text, click **Create New Timestamp Marker** for timestamp data, or **Create New Data Marker** for value data. Enter a name for that location, which is now a marker. The ICU will save where the highlighted text is located on the page into a marker, and the marker will be stored in the configuration file. The user will associate a PI tag with a data marker in the tag configuration, which is described in section PI Point Configuration. These steps are described in further detail below.

There are two different types of markers: data and timestamp markers. Data markers are created to be the value that is stored for a PI point. Data markers can be cast into any of the supported PI data types (assuming the cast is legal; for example, casting "4kl23" to an integer is of course not legal). This includes int16, int32, float16, float32, float64, string, and digital. Timestamp markers are markers created to be the timestamps for the data markers. Each data marker requires a timestamp source. This can either be a defined timestamp marker, or the data marker can use the current clock time as its timestamp, if no timestamp is available on the actual HTML page. Timestamp markers may be in many different date/time formats.

By selecting the markers in the **Data Markers** field in the ICU window, you can see with which timestamp markers those data markers are associated. This association cannot be changed from this screen; it can only be viewed.

Creating New Markers

1. Make sure the desired configuration file is selected in the **Current Configuration File** field. This is the file in which the locator script will be held. The ellipsis button will allow you to browse for a file. If a non-existing file is selected, a dialog box appears prompting you to create a new configuration file.
2. Click **Edit Markers** to open a web browser screen that shows all the data and timestamp marker information. The following is what the upper-right corner of the Edit Markers page looks like:



3. To create a new marker, highlight the location of the data on the web browser window that appears, and click **Create New Data Marker** or **Create New Timestamp Marker** to make a new marker of the respective type (hovering your mouse over the buttons shown above will reveal their function).

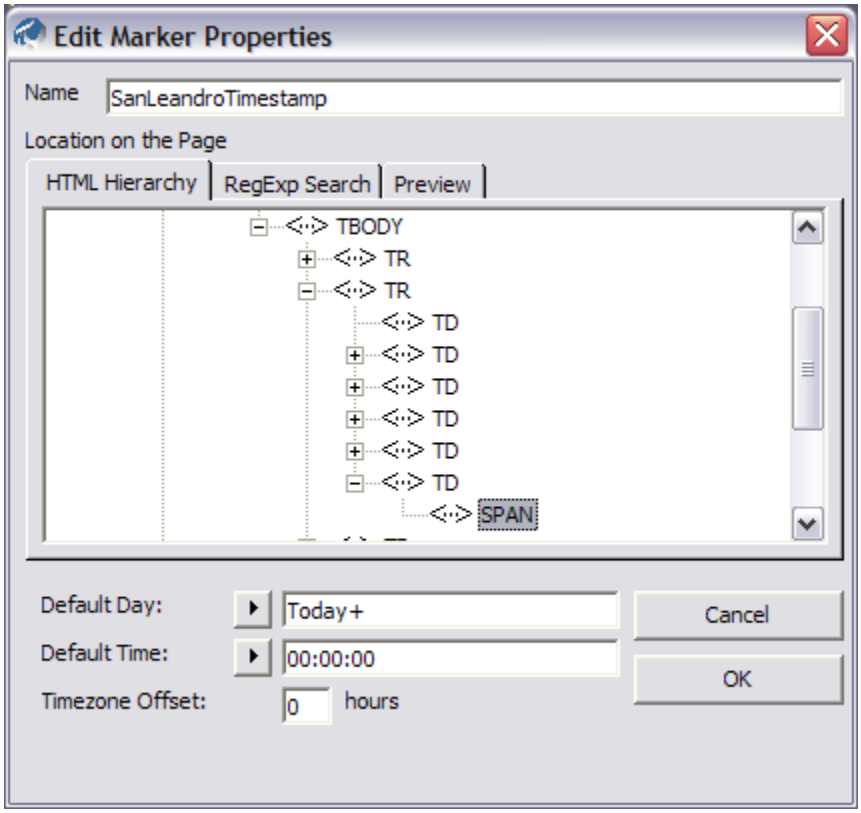
Editing Markers

When a new marker is created, or when you click **Edit Selected Timestamp** (or **Data**) **Marker** after selecting a marker in one of the two lists, the properties window will appear for the new (in the case of a new marker) or selected (in the case of an already-existing marker) marker.

The name field at the top of the page lets you specify an identifier to give this marker. This name should be unique. For data markers, this name is the name that will be specified in the InstrumentTag PI point attribute to associate the PI point with the data marker. For timestamp markers, this name is the name that will be selected in the edit window of a data marker, for associating a timestamp marker with a data marker.

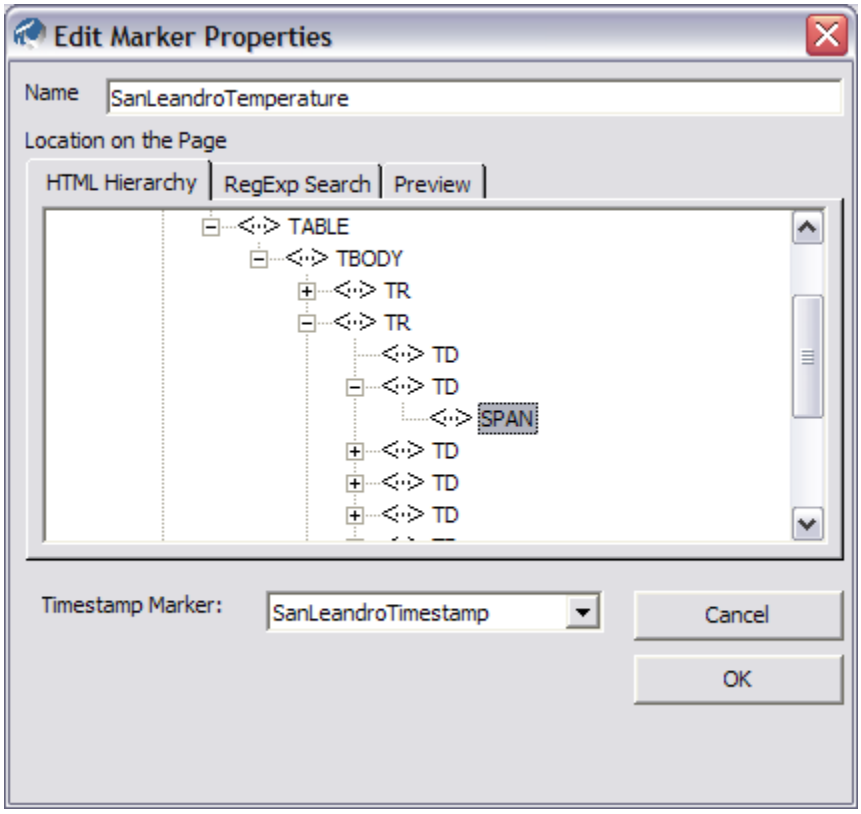
1. Select a new name for the marker. Make sure it is unique (per XML configuration file). Type that into the **Name** field.

2. If this is a timestamp marker, the **Edit Marker Properties** window will look like this:



- **Default Day** is telling the interface how it should handle situations where there is a time without a date. Click on the arrow button next to the text field to make a selection. `Today` will add the current local date to the time that was read on the HTML page. `Yesterday` will take today's date, subtract a day, and add that to the time. `Today` with extra logic will use the current date, but in the case that the combination of date plus time results in a timestamp that is more than 10 minutes into the future, it will subtract a day. This is useful in cases where, for example, the interface reads a page at 12:01 am, but the time on the page says 11:59 pm. If the current day's date were to be used, the timestamp would be today at 11:59 pm, when in reality, the desired date would be yesterday's date. Hardcoded will allow you to specify a hard-coded date, in PI date format (dd-mmm-yy or dd-mmm-yyyy). This is not very useful for normal operation of the interface, but can be useful if you need to read old pages that did not have dates on them, only times.
- **Default Time** instructs the interface how it should handle situations where there is a date without a time. This time will be applied to the date.
- **Timezone Offset** is a floating point number that instructs the interface how many hours to add to the timestamp (or subtract if it is a negative number). For example, for an interface running on US Pacific Time (GMT-8), but reading data from US Eastern time (GMT-5), this number should be -3.

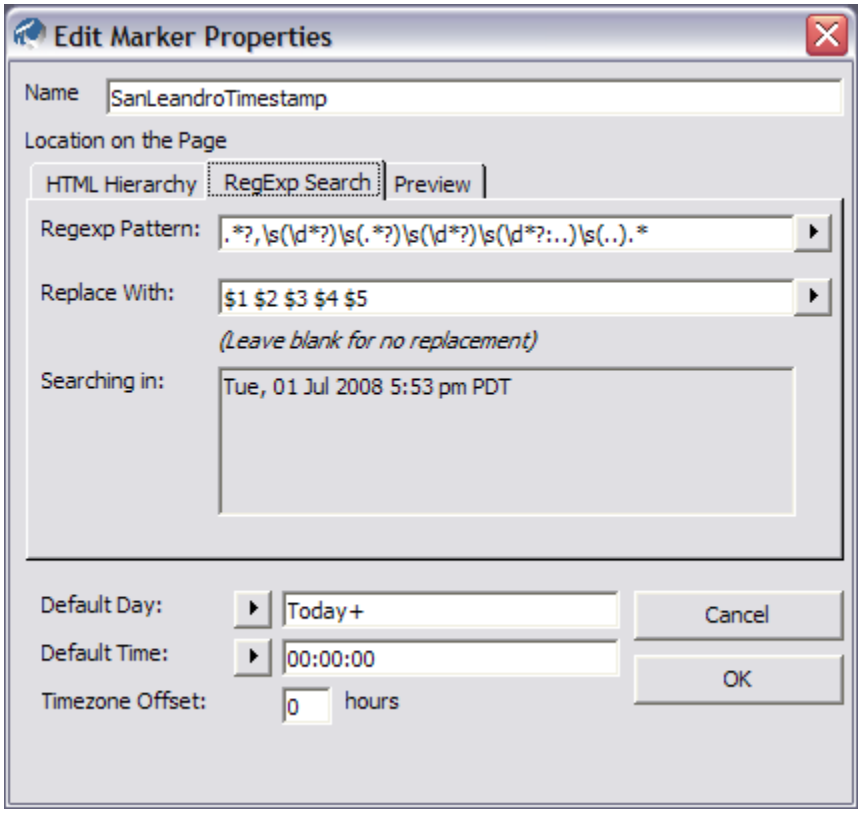
3. If this is a data marker, the **Edit Marker Properties** window will look like this.



   - **Timestamp Marker** is a dropdown box that lists all the timestamp markers that have been created so far. Also listed is [Use Current Timestamp]. That option will set this data marker to use the current interface time as opposed to reading a timestamp off the HTML page.

4. The **HTML Hierarchy** tab lets you select which node in the HTML hierarchy will be read for data. This tree view is a representation of how Internet Explorer exposes the HTML page to the interface. This box shows how you tell the interface where your data is. Normally, you do not have to edit this tree view box, because the correct node was selected when you initially created the marker. So edit this box if you really know what you are doing.

5. Clicking **RegExp Search** tab displays the following:



The text inside the node selected in the **HTML Hierarchy** tab may not be exactly the text you want to store into a PI point. See section Pattern Matching for an example. The regexp search and replace functionality lets you find the exact text you're looking for. Refer to the *Regular Expressions Tutorial* for detailed information and many examples on how to use regexp to get the correct data out of your HTML page. A quick summary follows.

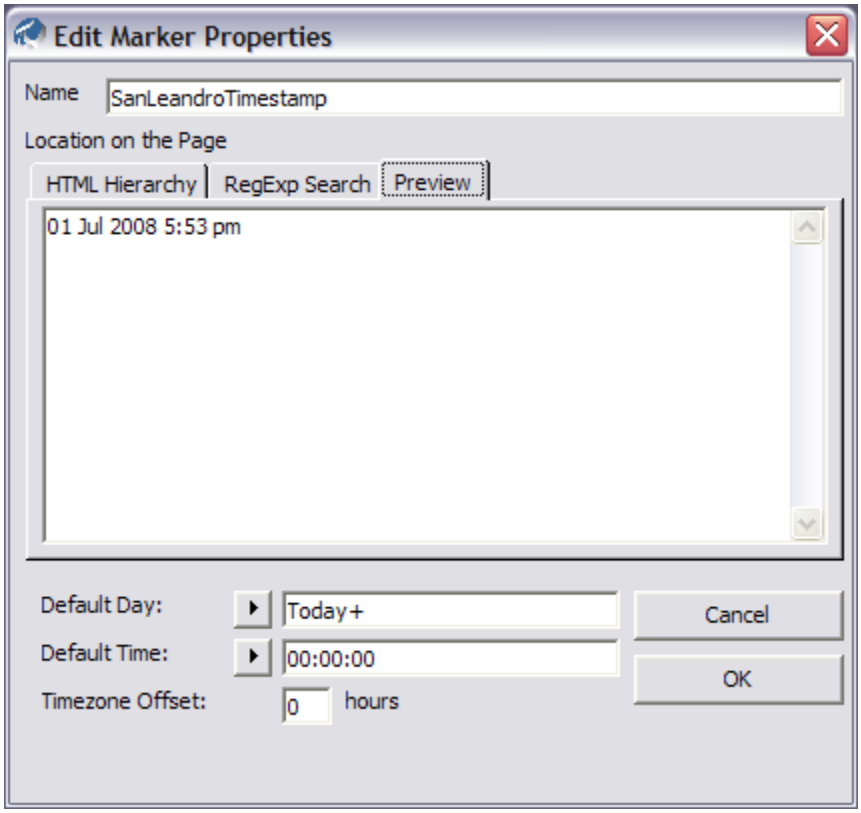
Click the **Preview** tab to see what text would be selected if the regexp fields were not changed. There is a good chance that the data you were looking for has already been found without using the regexp fields. In the example HTML in section Pattern Matching, the data values for temperature, humidity, and barometric pressure should be found correctly, because the numbers are alone inside the HTML tags. However, the timestamp marker will not be read correctly, because there is additional data inside the HTML tags besides the timestamp. If you were configuring the timestamp marker for that page, you would find "Weather data for December 12, 2001 12:32 pm" shown in the preview frame. The goal is to narrow this text down to a date.

First, delete the old pattern (".*"). Then specify patterns for the data you want, and the data you do not want. You want to remove the "Data for" part, but keep the following date part. One pattern you could use is `.*for\s.*`, (period-asterisk-f-o-r-backslash-s-period-asterisk.) The first period-asterisk is a wildcard search of any number of characters. The f-o-r matches the "for" in "Data for." The backslash-s matches the space after "for". The final period-asterisk matches the actual date part. What we want to keep in this example is the date part, which is represented in the pattern by the last period-asterisk. So place that part in parentheses, like this: `.*for\s(.*)`

Use parentheses to create a group in the pattern. In the *Regular Expressions Tutorial*, groups are discussed in more detail. As discussed in the tutorial, there can be several groups defined in a pattern. In our case, there is only one group defined. We want to select that first (and only) group as our final data, so select (found 1) from the menu that appears when you click on the arrow button next to the **Replace With** field. Then, when you click the **Preview** tab, only the date appears.

Pattern matching and substitution can be complicated, but typically the HTML page you want to read will not be formatted in a way that would require you to use anything other than the default. Otherwise, read the *Regular Expressions Tutorial*.

6. The **Preview** tab shows you what the results of your selection (along with any changes you made in the **RegExp Search** tab you made) are. This tab is how the marker will be interpreted by the interface.



Validate Markers

Since version 2.0, the HTML interface uses a third-party library called Curl to perform the download of all web pages. However, in order to facilitate easier configuration of the interface, the ICU (and HTMLConfigUtil) uses Internet Explorer as its method of getting the web pages off the internet. Sometimes this causes differences in what the user has configured and what the HTML interface sees as its target web page.

Click **Validate Markers** to display a screen that shows what the interface would see when it attempts to navigate to the target page and search for the markers on that page. This is a good way to test to make sure your markers will be properly read by the HTML interface after the configuration is finished.

If the markers do not appear correctly, there are a few techniques for troubleshooting.

Misc

Click this button to open a dialog box with other options, mostly used to modify how the interface runs. These options correspond to command line parameters discussed above in section Command-line Parameters.

Additional Parameters

This section is provided for backwards compatibility. If, for some reason, there are additional parameters required for a newer version of the interface to operate, and the `html.dll` file that is available on the current computer is not up to date, the ICU will not be able to correctly configure the newly added parameters. Use the **Additional Parameters** field to enter options that are not available in the graphical part of the ICU. This text box is normally left blank unless the versions of `html.ocx` and the PI Interface for HTML executable file are out of sync.

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

Configuring the Interface Without the PI ICU

For communicating with PI Data Archives earlier than version 3.3, the PI ICU cannot be used. A small tool (`HTMLConfigUtil.exe`) has been supplied for those without the PI ICU. The GUI is almost exactly the same as if you were using the PI ICU, but much of the functionality of the PI ICU is not available. For example, this utility cannot edit a startup `.bat` file, so you will be responsible for maintaining that. The options for configuring the startup file are in section Startup Command File.

An XML configuration file can be edited in just the same way from this tool as from the PI ICU.

Command-line Parameters

Any options discussed below marked with an asterisk before the name of the parameter are not normally used, and thus are not configurable by using the PI ICU except by manually typing the parameter in the **Additional Parameters** field of the **html** tab in the PI ICU. If you use more than one in the **Additional Parameters** field, separate each command-line parameter with a space.

Parameter	Description
/db=# Optional	Use to print out debug messages. The value of this flag is determined by adding the number that accompanies the debugging messages you want to see that are listed below: 1 – Reading the XML configuration file. 2 – Adding points to the interface's internal list and finding their corresponding data markers in the configuration file. 4 – Connecting to the HTML page server (if there is one) and downloading the HTML pages. 8 – Parsing the HTML into a tree-like hierarchy. 16 – Writing data to PI points. 32 – Taking the text from a marker and converting it to the appropriate type (timestamp for timestamp markers, or numeric for data markers) 64 – Generate `curldebug.log` file for debug messages printed by libCurl.
/dltimeout=# Optional	Use to indicate how long (in seconds) the interface should wait for your page or pages to download before timing out. The default is 60 seconds.
/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.
/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

Parameter	Description
	the second scan class, and so on. Two scan classes are defined in the following example: **/f=00:01:00,00:00:05 /f=00:00:07** or, equivalently: **/f=60,5 /f=7** The first scan class has a scanning frequency of 1 minute with an offset of 5 seconds, and the second scan class has a scanning frequency of 7 seconds. When an offset is specified, the scans occur at discrete moments in time according to the formula: scan times = (reference time) + n(frequency) + offset where n is an integer and the reference time is midnight on the day that the interface was started. In the above example, frequency is 60 seconds and offset is 5 seconds for the first scan class. This means that if the interface was started at 05:06:06, the first scan would be at **05:07:05**, the second scan would be at **05:08:05**, and so on. Since no offset is specified for the second scan class, the absolute scan times are undefined. The definition of a scan class does not guarantee that the associated points will be scanned at the given frequency. If the interface is under a large load, then some scans may occur late or be skipped entirely. See the section "Performance Summaries" in *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.
/htmlconfigfile=<UNC Path> Required	Use to specify the XML file that contains the information configured by the Interface-Specific Parameters tab on the PI ICU. For example, **/htmlconfigfile=d:\pipc\Interfaces\HTML** html1config.xml This file is created by the PI ICU, or by the simple PI Interface for HTML configuration tool.

Parameter	Description
/host=*host:port* Required	The **/host** parameter is used to specify the PI Data Archive node. *Host* is the IP address of the PI Data Archive node or the domain name of the PI Data Archive node. *Port* is the port number for TCP/IP communication. The *port* is always 5450. It is recommended to explicitly define the host and port on the command-line with the **/host** parameter. Nevertheless, if either the host or port is not specified, the interface will attempt to use defaults. Examples: The interface is running on an interface node, the domain name of the PI Data Archive node is Marvin, and the IP address of Marvin is 206.79.198.30. Valid **/host** parameters would be: **/host=marvin** **/host=marvin:5450** **/host=206.79.198.30** **/host=206.79.198.30:5450**
/id=*x* Highly Recommended	The **/id** parameter is used to specify the interface identifier. The interface identifier is a string that is no longer than 9 characters in length. UniInt concatenates this string to the header that is used to identify error messages as belonging to a particular interface. See 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**
/outputhtml=(Y)es/(N)o Optional	Used for debugging purposes. If set to yes (or y), the interface will write out all final HTML pages it receives. This does not include pages received in the process of getting to the final HTML page. The interface writes these pages to the directory of the interface executable the form `HTML_retrieved_yyyymmddhhmmss.html`.
/parsetimeout=# Optional	Indicates how long (in seconds) the interface should wait for your page or pages to be parsed by Internet Explorer before timing out. The default is 60 seconds. This option is useful if you see a lot of parse timeout messages in the `pipc.log` file. Otherwise, leave the default value.
/plugin=<UNC path> Optional	Specifies the progid of a plug-in that should be used for dynamic URL generation and timestamp and value post-processing. When a plug-in is selected, the plug-in will be used to help navigation when configuring the HTML locator script. It will also be used when showing a preview of the data when configuring data and timestamp markers. If no plug-in is selected, no plug-in will be used in the configuration or in the execution of the interface. See section Appendix C on Plug-in Registration and Categorization.

Parameter	Description
/ps=*x* Required	The **/ps** parameter specifies the point source for the interface. *X* is not case sensitive and can be any <single/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 earlier than 1.6.x or the PI Data Archive version is earlier than 3.4.370.x, the PointSource is limited to a single character unless the SDK is being used.
/replace=(Y)es/(N) Optional	Instructs the interface to use archive replace calls instead of put snapshot calls to send the data to PI points. This is useful when the value for a given timestamp may change. The default value is no.
/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 Data Archive, *digstate* must be in the system digital state table. UniInt will use the first occurrence of *digstate* found in the table. If the **/stopstat** parameter is present on the startup command line, then the digital state `Intf Shut` will be written to each PI point when the interface is stopped. If neither **/stopstat** nor **/stopstat=***digstate* is specified on the command line, then no digital states will be written when the interface is shut down. Examples: **/stopstat=***shutdown* **/stopstat="***Intf Shut***"** The entire *digstate* value must be enclosed within double quotes when there is a space in *digstate*.
/suppresserrors=(Y)es/(N)o Optional	Instructs the interface not to send digital state errors to PI points. For example, in normal operation, if a marker for a tag cannot be found on the target HTML page, that tag will receive a CONFIGURE digital state. With this option set to yes, a message will be written to the log file, but the digital state will not be sent to the PI point. The default value is no.
/useragent=<string> Optional	Allows the interface to identify itself to the remote web server as a different web browser. Some web sites will return a different page for different browsers. A common user-agent string to use to mimic Internet Explorer 5.5 on Windows 2000 is "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)".

Sample PIHTML.bat File

The following is an example file:

```
REM=================================================================
REM
REM PIHTML.bat
REM
REM Sample startup file for the PI Interface for HTML
REM
REM=================================================================
REM
REM OSIsoft strongly recommends using PI ICU to modify startup files.
REM
REM  Sample command line
REM
     .\PIHTML.exe 1 ^
     /htmlconfigfile=.\PIHTMLExampleConfig.xml ^
     /db=0 ^
     /dltimeout=60 ^
     /parsetimeout=60 ^
     /suppresserrors=N ^
     /outputhtml=N ^
     /replace=N ^
     /PS=HTML ^
     /ID=1 ^
     /host=XXXXXX:5450 ^
     /f=00:30:00
REM
REM End of PIHTML.bat File
```

Converting Older Configuration Files

If you are using an XML configuration file from before version 1.2, you need to convert it to the new format before you can use the PI Interface for HTML. Use the utility included in the interface directory called `InterfaceConfigDocConverter.exe`. It transforms the XML configuration document.

> **Note:** Since the proxy server handling is different in the new version of the interface, all proxy server information will be lost and will need to be recreated.

Use the converter by running it on a command line, passing the path to old file as the first parameter, and the path to what you want your new file to be called as the second parameter:

```
InterfaceConfigDocConverter.exe
c:\PIPC\Interfaces\HTML\myoldconfig.xml
c:\PIPC\Interfaces\HTML\mynewconfig.xml
```

Starting with version 2.3.0.0, the interface encrypts Proxy and HTTP Security passwords. If you are using an XML configuration file from earlier versions (2.0 or 2.2) and it contains Proxy or/and HTTP Security passwords, you need to recreate the HTML Locator Script configuration item and reenter the passwords before you can use PI Interface for HTML. See HTML Locator Script section for technical details about Locator Script configuration item.

Security Note: For most protocols, the communications between the interface and a source web site are visible to a malicious eavesdropper, which can include the user IDs and passwords used to connect to the web sites. If the target website requires a password, https or VPN should be used to protect the password on the wire. Also, restrict access using permissions and enable security auditing for all access to the configuration 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 cold failover.

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

When configured for UniInt failover, the interface routes all PI point data through a state machine. The state machine determines whether to queue data or send it directly to a PI point depending on the current state of the interface. When the interface is in the active state, data sent through the interface gets routed directly to a PI point. 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 Data Archive 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)

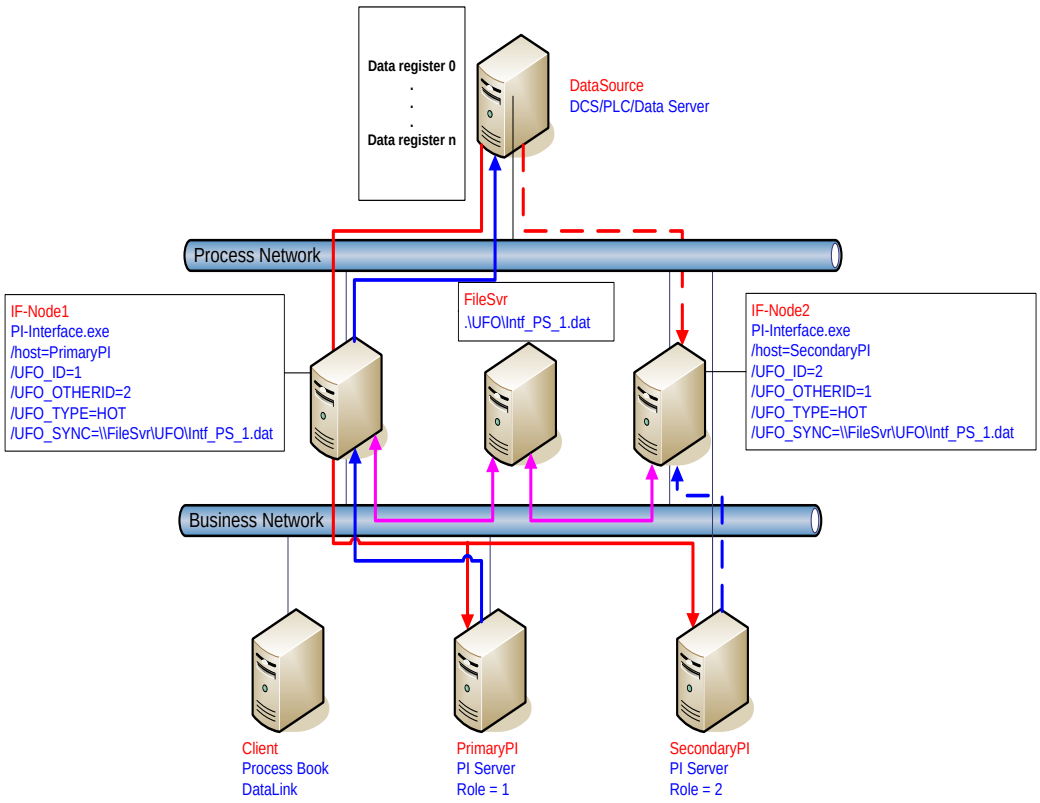


Figure 1: Synchronization through a Shared File (Phase 2) Failover Architecture

The Phase 2 failover architecture is shown in Figure 1 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 Data collective, you must point the primary and secondary interfaces to different members of the PI Data 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 Data Archive using available PI tools. • The **Active ID** control tag on the PI Data Archive must be set to the value of the running copy of the interface as defined by the **`/UFO_ID`** startup command-line parameter. • The **Heartbeat** control tag on the PI Data Archive must be changing values at a rate specified by the **`/UFO_Interval`** startup command-line parameter. 4. Stop the primary interface. 5. Start the backup interface interactively without buffering. Notice that this copy will become the primary because the other copy is stopped. 6. Repeat steps 2, 3, and 4. 7. Stop the backup interface. 8. Start buffering. 9. Start the primary interface interactively. 10. Once the primary interface has successfully started and is collecting data, start the backup interface interactively. 11. Verify that both copies of the interface are running in a failover configuration. • Review the `pipc.log` file for the copy of the interface that was started first. The log file will contain messages that indicate the failover state of the interface. The state of this interface must have changed as indicated with an informational message stating "`UniInt failover: Interface in the "Primary" state and actively sending data to PI. Backup interface available.`" If the interface has not changed to this state, browse the log file for error messages. For details relating to informational and error messages, refer to the Messages section below. • Review the `pipc.log` file for the copy of the interface that was started last. The log file will contain messages that indicate the failover state of the interface. A successful start of the interface will be indicated by an informational message stating "`UniInt failover: Interface in the "Backup" state.`" If the interface has failed to start, an error message will appear in the log file. For details relating to informational and error messages, refer to the Messages section below. 12. Verify data on the PI Data Archive using available PI tools. • The **Active ID** control tag on the PI Data Archive must be set to the value of the running copy of the interface that was started first as defined by the **`/UFO_ID`** startup command-line parameter. • The **Heartbeat** control tags for both copies of the interface on the PI

Step	Description
	Data Archive must be changing values at a rate specified by the **/UFO_Interval** startup command-line parameter or the scan class which the points have been built against. 13. Test Failover by stopping the primary interface. 14. Verify the backup interface has assumed the role of primary by searching the `pipc.log` file for a message indicating the backup interface has changed to the "`UniInt failover: Interface in the "Primary" state and actively sending data to PI. Backup interface not available`." The backup interface is now considered primary and the previous primary interface is now backup. 15. Verify no loss of data in the PI Data Archive. There may be an overlap of data due to the queuing of data. For cold failover, there may be a short 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 Data Archive connection for a single interface copy). UniInt hot failover guarantees no data loss with a single point of failure; verify no data loss by checking the data in the PI Data Archive and on the data source. For warm and cold failover, short data gaps are expected. 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 Data Archive for exceptions and PI point updates The value of the **/Host** startup parameter depends on the PI Data Archive configuration. If the PI Data Archive is not part of a PI Data collective, the value of **/Host** must be identical on both interface computers. If the redundant interfaces are being configured to send data to a PI Data collective, the value of the **/Host** parameters on the different interface nodes should equal to different members of the PI Data collective. This parameter ensures that outputs continue to be sent to the data source if one of the PI Data Archives becomes unavailable for any reason.	For IF-Node1 PrimaryPI / **None** For IF-Node2 SecondaryPI / **None**

Failover Control Points

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

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

Point	Description	Value / Default
ActiveID	Monitored by the interfaces to determine which interface is currently sending data to the PI Data Archive. **ActiveID** must be initialized so that when the interfaces read it for the first time, it is not in an error state. **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.

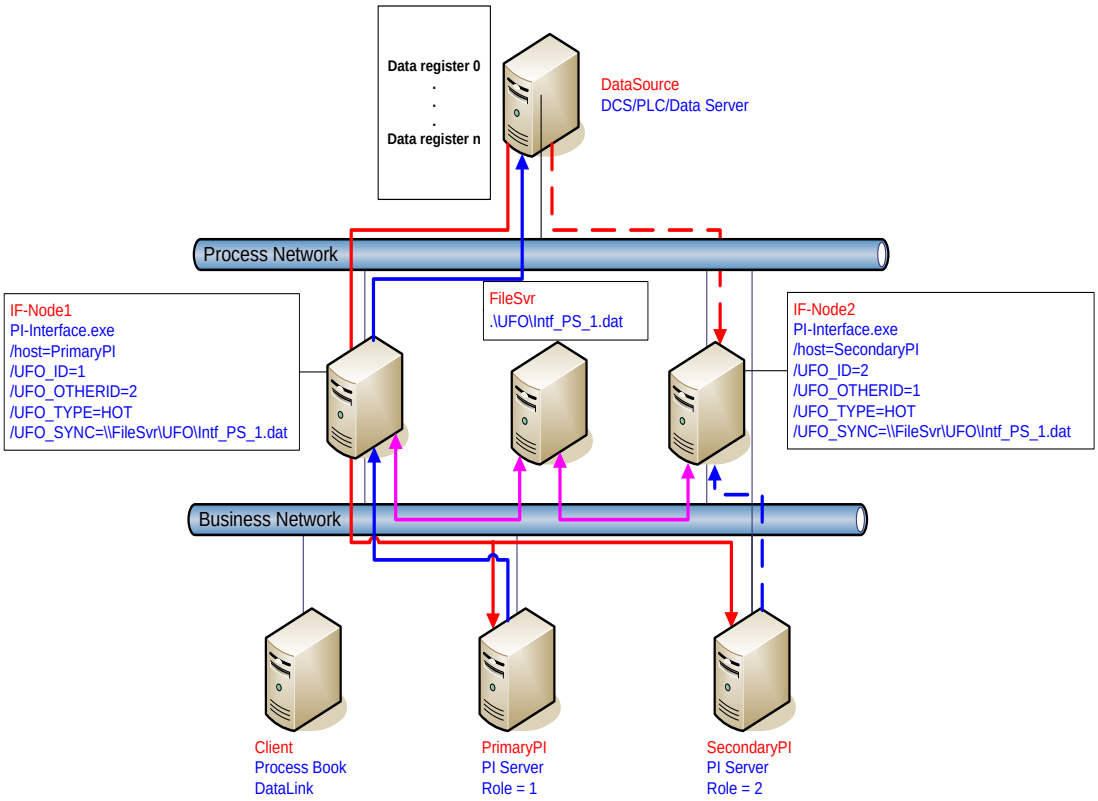
PI Tag ExDesc	Required / Optional	Description	Value
		interface is running but cannot communicate with the data source. 2 = Backup No PI Connection: The interface is running and connected to the data source but has lost its communication to the PI Data Archive. 3 = Backup: The interface is running and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

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

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

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

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



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

The major difference between synchronizing the interfaces through the data source (Phase 1) and synchronizing the interfaces through the shared file (Phase 2) is where the control data is located. When synchronizing through the data source, the control data is acquired directly

from the data source. We assume that if the primary interface cannot read the failover control points, then it cannot read any other data. There is no need for a backup communications path between the control data and the interface.

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

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

The primary interface also monitors the connection with the shared file to maintain the integrity of the failover configuration. If the primary interface can read and write to the shared file with no errors but the backup control information is not changing, then the backup is experiencing some error condition. To determine exactly where the problem exists, the primary interface uses the path to the PI Data Archive to establish the status of the backup interface. For example, if the backup interface information indicates 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 the PI Data Archive to determine if one or the other is not updating the shared file and why.

Steady State Operation

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

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

The interaction between the interface and the shared file is fundamental to failover. The discussion that follows only refers to the data written to the shared file. However, every value written to the shared file is echoed to the tags on the PI Data Archive. Updating of the tags on the PI Data Archive is assumed to take place unless communication with the PI Data Archive

is interrupted. The updates to the PI Data Archive will be buffered by bufserv or BufSS in this case.

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

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

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

As previously mentioned, in a hot failover configuration the backup interface actively collects data but does not send its data to PI points. To eliminate any data loss during a failover, the backup interface queues data in memory for three failover update intervals. The data in the queue is continuously updated to contain the most recent data. Data older than three update intervals is discarded if the primary interface is in a good status as determined by the backup. If the backup interface transitions to the primary, it will have data in its queue to send to the PI point. This queued data is sent to the PI points using the same function calls that would have been used had the interface been in a primary state when the function call was received from UniInt. If UniInt receives data without a timestamp, the primary copy uses the current PI Data Archive time to timestamp data sent to PI points. Likewise, the backup copy timestamps data it receives without a timestamp with the current PI Data Archive 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 2 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.

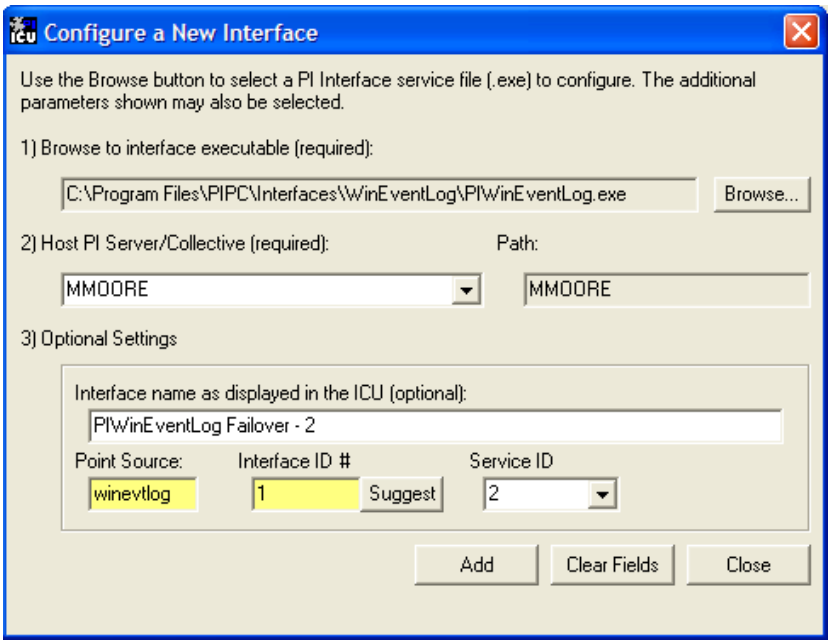


Figure 2: PI ICU configuration screen shows that the "PS/ID combo is already in use by the interface." The user must ignore the yellow boxes, which indicate errors, and click the *Add* button to configure the interface for failover.

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

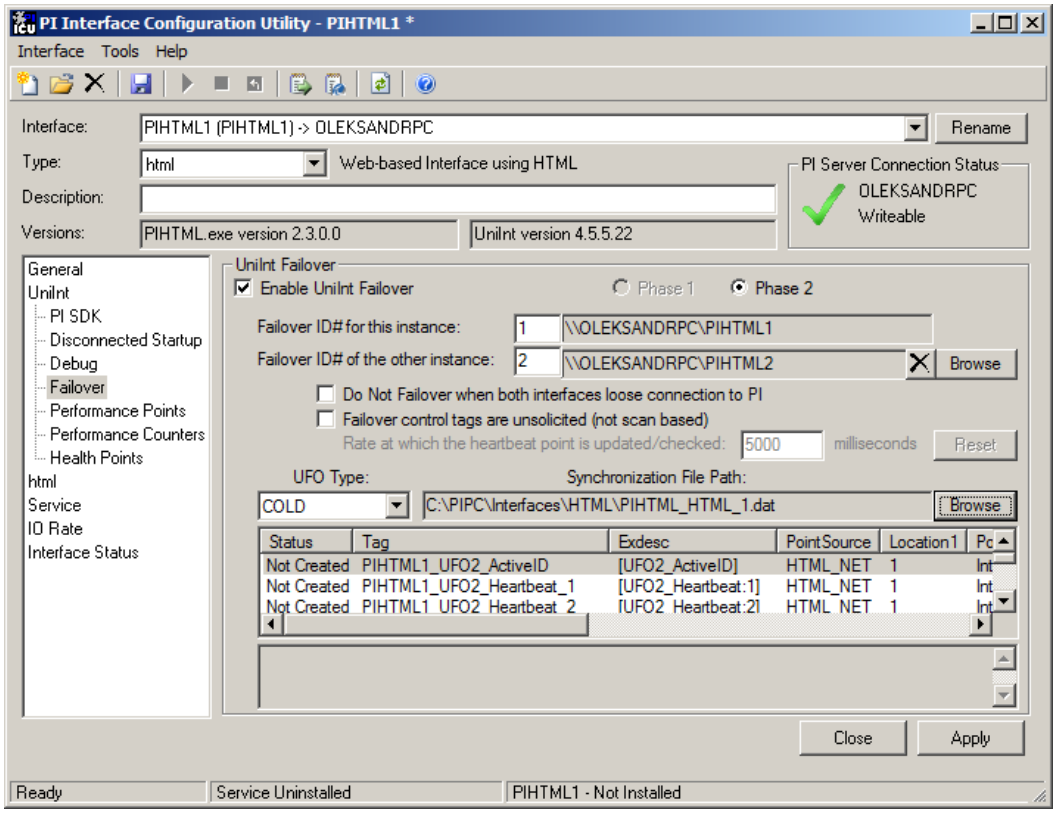


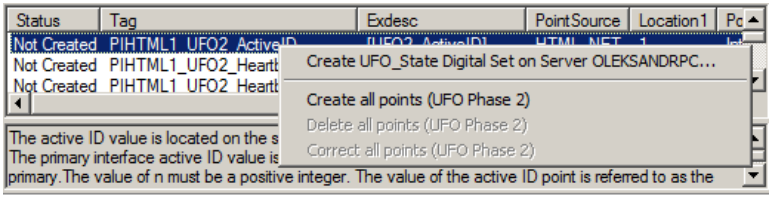
Figure 3: The figure above illustrates the PI ICU failover configuration screen showing the UniInt failover startup parameters (Phase 2). This copy of the interface defines its Failover ID as 2 (**/UFO_ID=2**) and the other Interfaces Failover ID as 1 (**/UFO_OtherID=1**). The other failover interface 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 later) or the Digital States plug-in in the SMT 3 Utility (3.0.0.7 or later).

Using the PI ICU Utility to create Digital State Set

To use the UniInt *Failover* page to create the UFO_State digital state set, right-click on any of the failover tags in the tag list and then click the *Create UFO_State Digital Set on PI Data Archive XXXXXX...* command, where *XXXXXX* is the PI Data Archive 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 Data Archive.

Using the PI SMT 3 Utility to create Digital State Set

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

The procedure below outlines the steps necessary to create a digital set on a PI Data Archive 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 Data Archive from the PI Data Archives window. If the desired PI Data Archive is not listed, add it using the PI Connection Manager. A view of the SMT application is shown in Figure 4 below.
3. From the *System Management Plug-Ins* window, expand *Points* then select *Digital States*. A list of available digital state sets will be displayed in the main window for the selected PI Data Archive. Refer to Figure 4 below.
4. In the main window, right-click on the desired PI Data Archive and select the *Import from File* command. Refer to Figure 4 below.

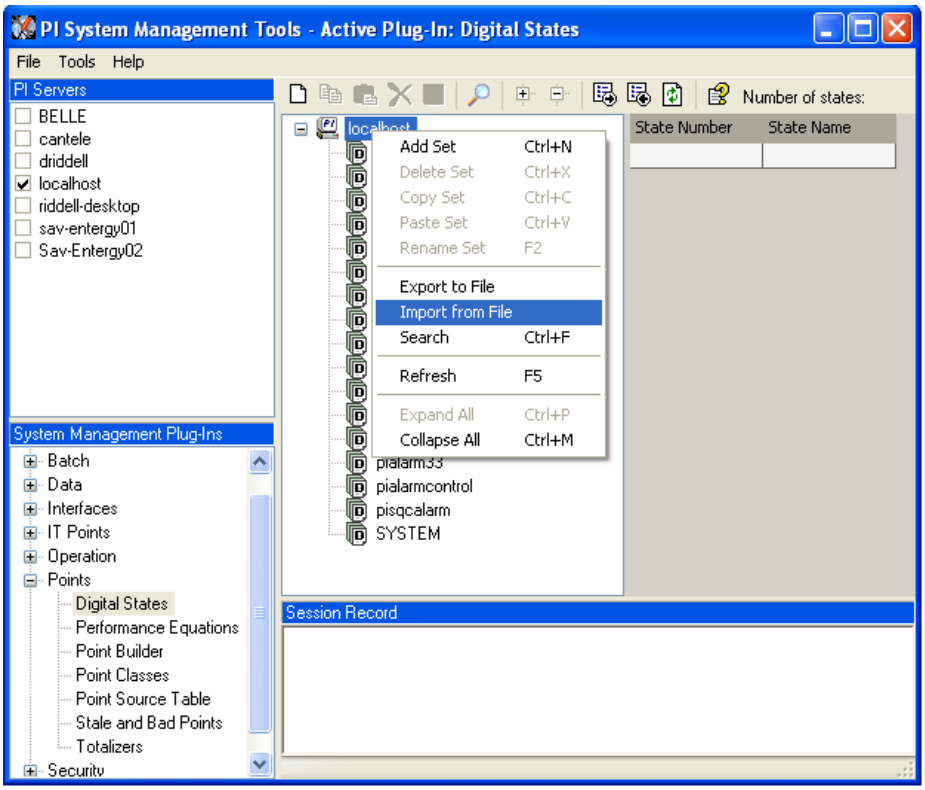


Figure 4: PI SMT application configured to import a digital state set file. The PI Data Archives window shows the "localhost" PI Data Archive selected along with the System Management Plug-Ins window showing the Digital States Plug-In as being selected. The digital state set file can now be imported by selecting the *Import from File* command.

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

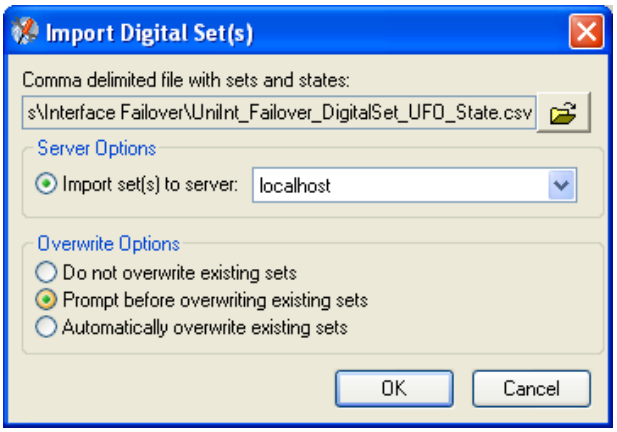


Figure 5: PI SMT application *Import Digital Set(s)* window. This view shows the `UniInt_Failover_DigitalSet_UFO_State.csv` file as being selected for import. Select the desired *Overwrite Options* by choosing the appropriate option button.

6. Click on the *OK* button. Refer to Figure 5 above.

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

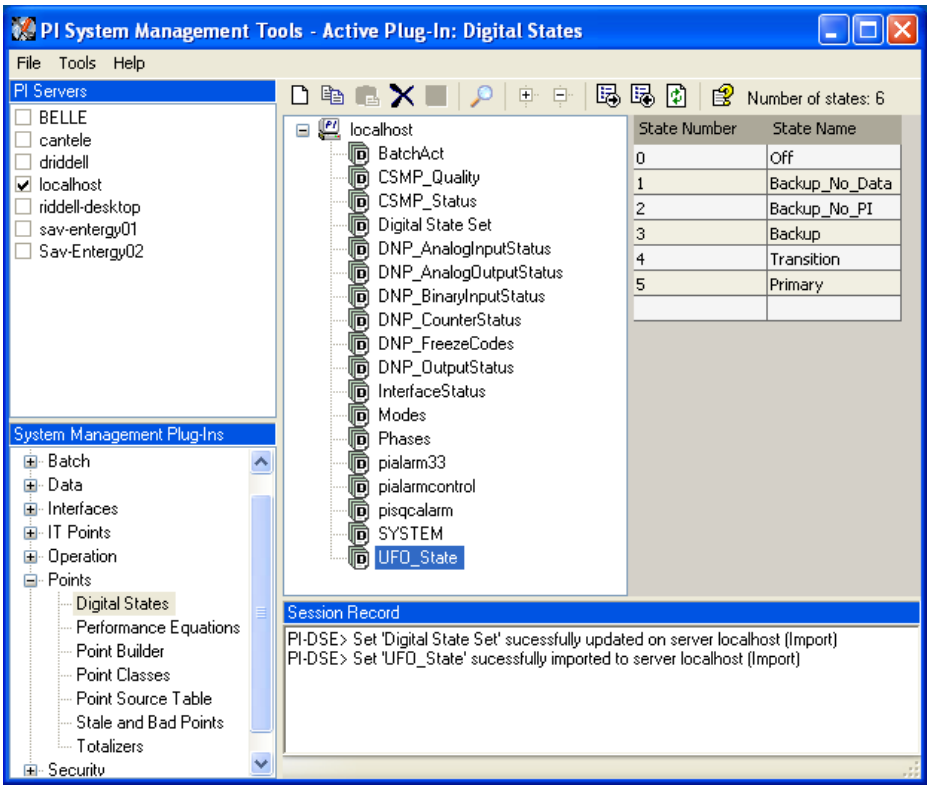


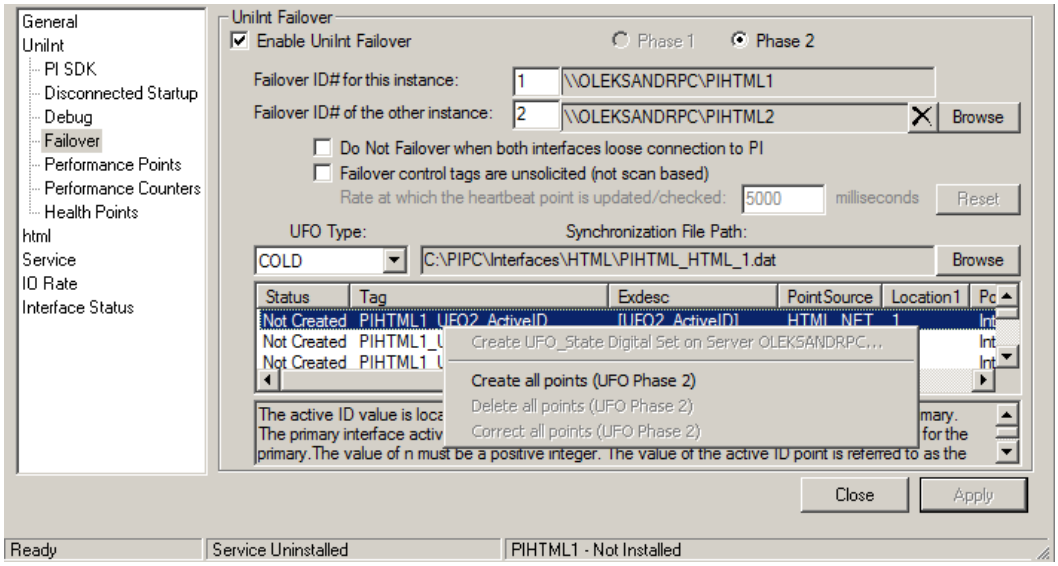
Figure 6: The PI SMT application showing the `UFO_State` digital set created on the "localhost" PI Data Archive.

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

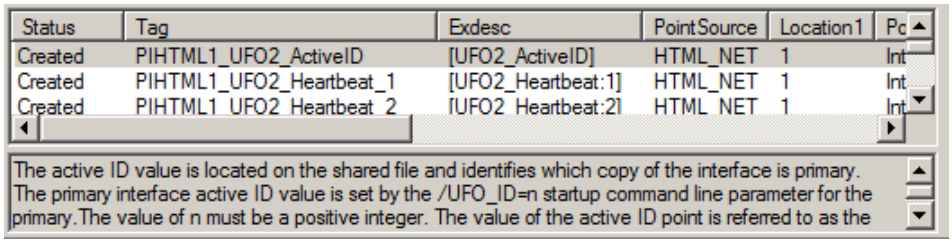
The ICU can be used to create the UniInt Failover Control and State Tags.

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 Data Archive yet. *Create UFO_State Digital Set on PI Data Archive 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.

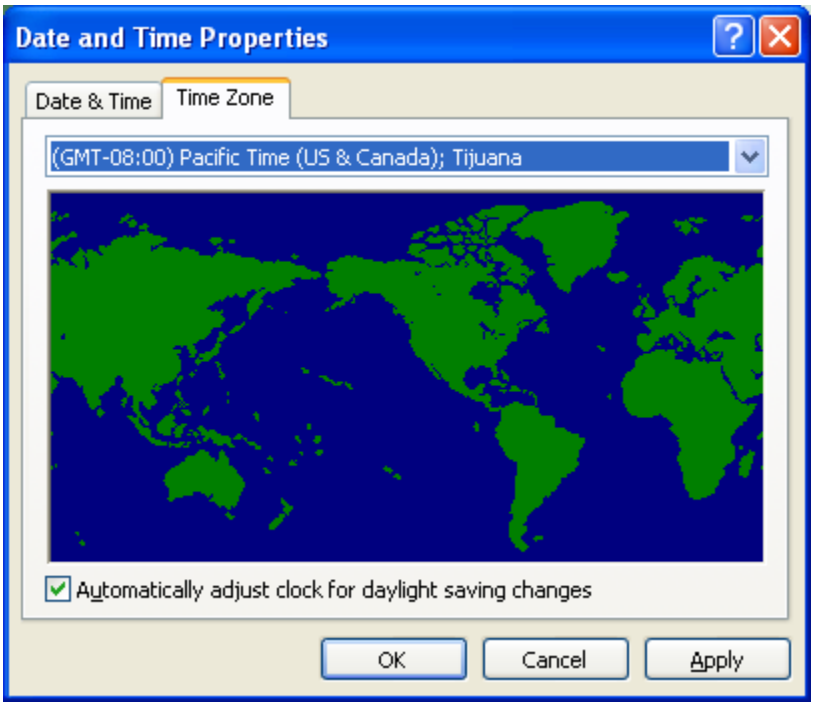


Once the failover control and failover state tags have been created the *Failover* page of the ICU should look similar to the illustration below.



Chapter 10. Interface Node Clock

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

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

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

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

Authentication

Interface instances are usually configured to run as Windows services. Since a service runs in a non-interactive context, a PI Trust is required to authenticate the interface service to the PI Data Archive.

PI Data Archive v3.3 and Higher

Security configuration using piconfig

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

```
C:\PI\adm> piconfig
@table pitrust
@mode create
@istr Trust,IPAddr,NetMask,PIUser
a_trust_name,192.168.100.11,255.255.255.255,piadmins
@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 identify, user, or group the interface is entrusted as

Security Configuring using Trust Editor

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

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

PI Data Archive v3.2

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

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

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

Authorization

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

Database Security	Permission	Notes
PIPOINT	r	

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

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

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

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

Chapter 12. Starting / Stopping the Interface

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

Starting Interface as a Service

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

```
PIHTML.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:

```
PIHTML.exe /stop
```

The service can be removed by:

```
PIHTML.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 Data Archives. OSIsoft strongly recommends that you enable buffering on your interface nodes. Otherwise, if the interface node stops communicating with the PI Data Archive, you lose the data that your interfaces collect.

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

If you have PI Data Archives that are part of a PI Data collective, PIBufss supports *n-way buffering*. N-way buffering refers to the ability of a buffering application to send the same data to each of the PI Data Archives in a PI Data collective. (Bufserv also supports n-way buffering, but OSIsoft recommends that you run PIBufss instead.)

Which Buffering Application to Use

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

You can use PIBufss only under the following conditions:

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

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

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

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

It is technically possible to run Bufserv on the PI Data Archive Node. However, OSIsoft does not recommend this configuration.

How Buffering Works

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

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

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

The buffering application (either Bufserv or PIBufss) in turn

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

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

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

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

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

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

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

Buffering and PI Data Archive Security

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

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

Buffering Application	Application Name field for PI Trust
PI Buffer Subsystem	PIBufss.exe

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

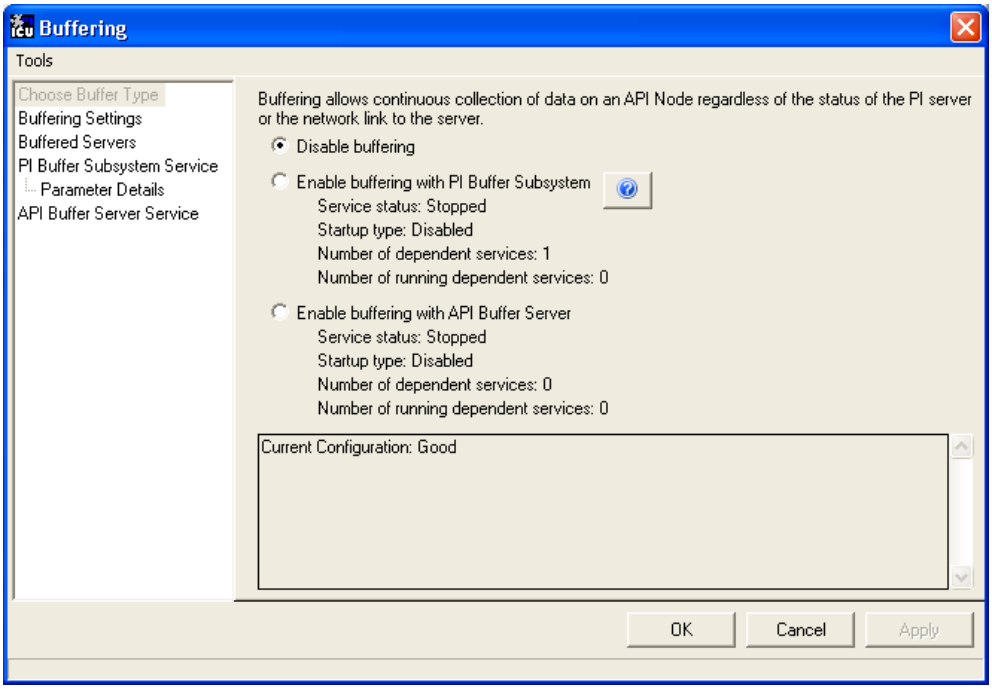
To use a process name greater than 4 characters in length for a trust application name, use the LONGAPPNAME=1 in the PIClient.ini file.

See the Security chapter for additional information.

Enabling Buffering on an Interface Node with the ICU

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

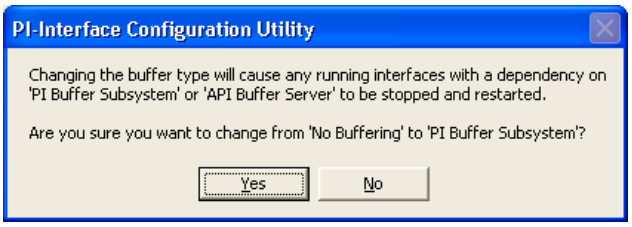
Choose Buffer Type



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

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

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

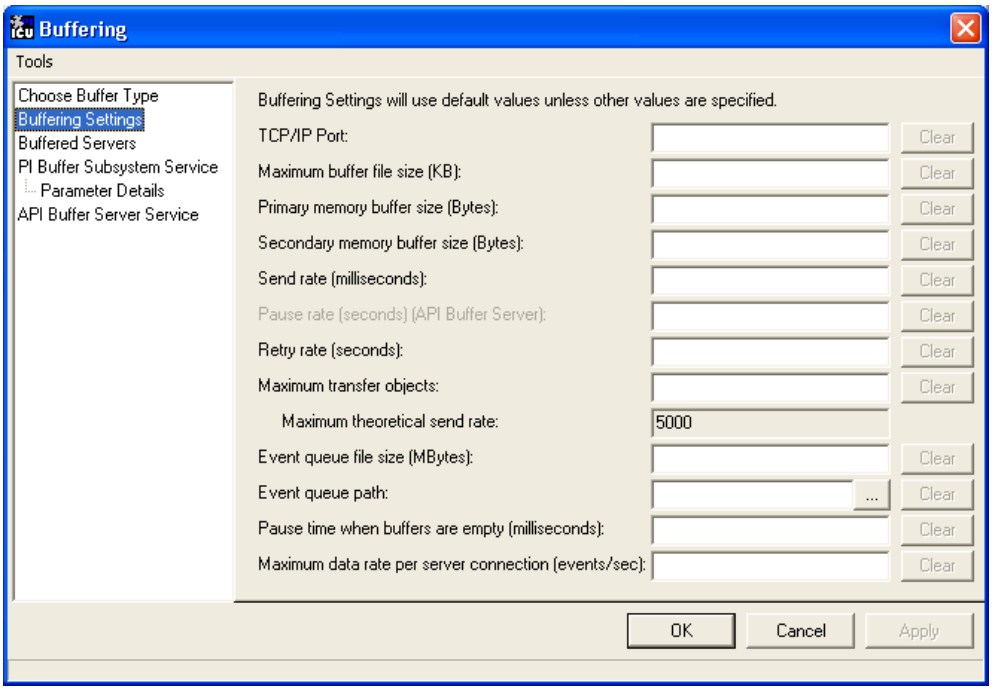


Buffering Settings

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

PIBufss

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



Primary and Secondary Memory Buffer Size (Bytes)

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

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

Send rate (milliseconds)

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

Maximum transfer objects

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

Event Queue File Size (Mbytes)

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

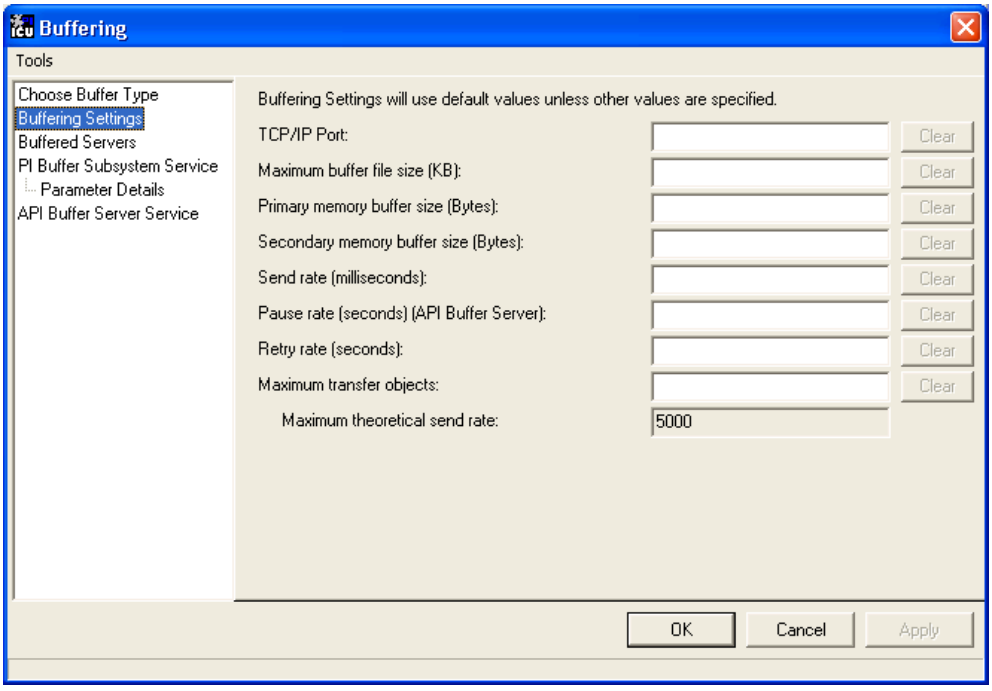
Event Queue Path

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

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

Bufserv

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



Maximum buffer file size (KB)

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

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

Primary and Secondary Memory Buffer Size (Bytes)

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

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

Send rate (milliseconds)

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

Maximum transfer objects

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

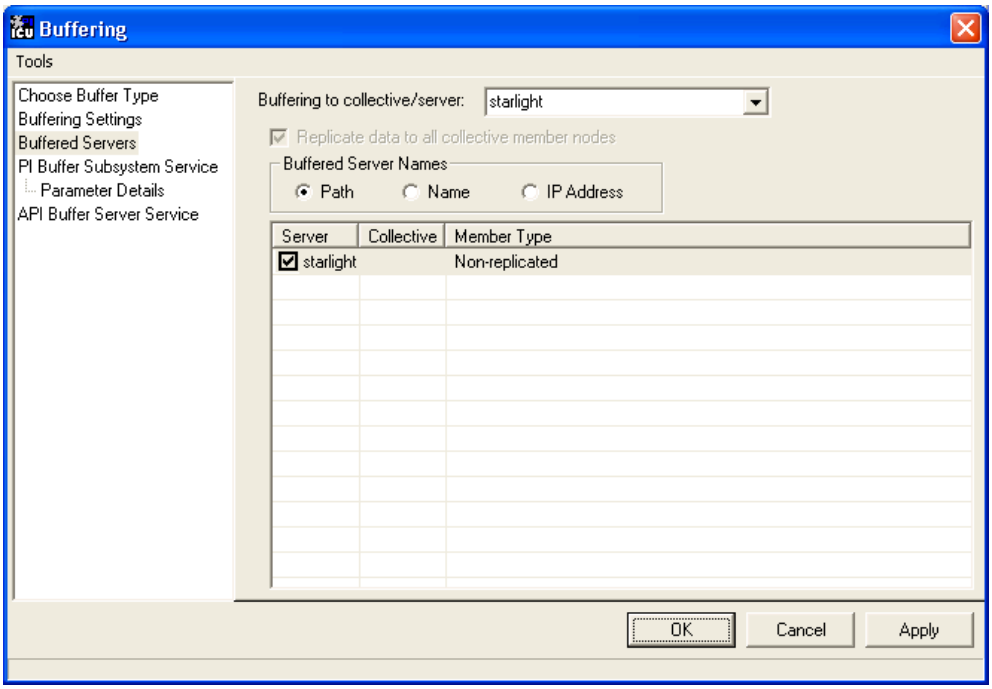
Buffered Servers

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

PIBufss

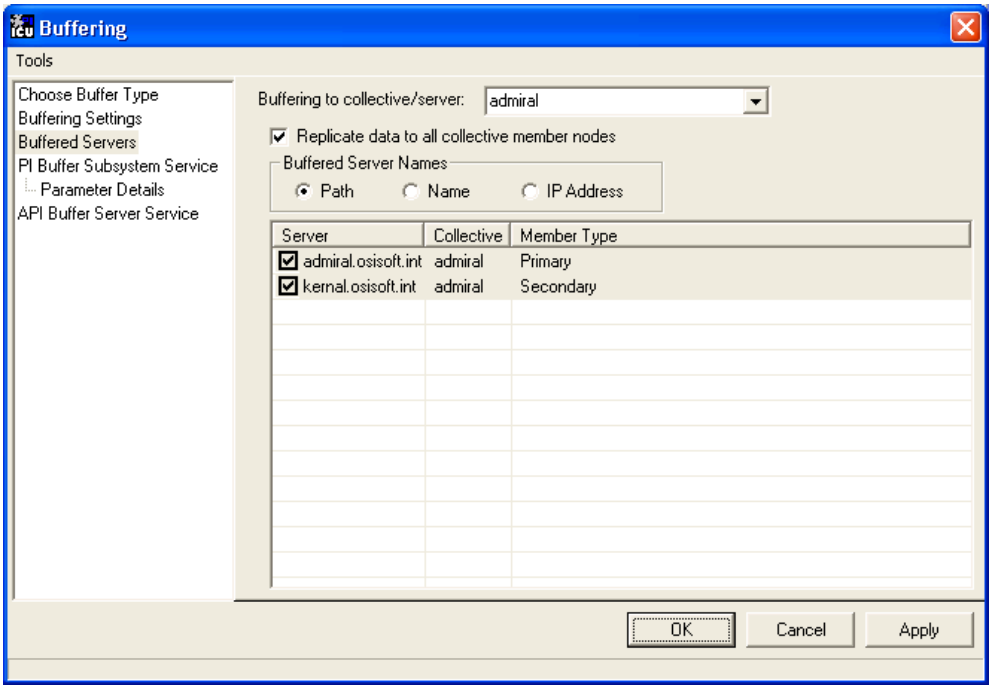
PIBufss buffers data only to a single PI Data Archive or a PI Data collective. Select the PI Data Archive or the PI Data collective from the *Buffering to PI Data collective/archive* drop down list box.

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



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

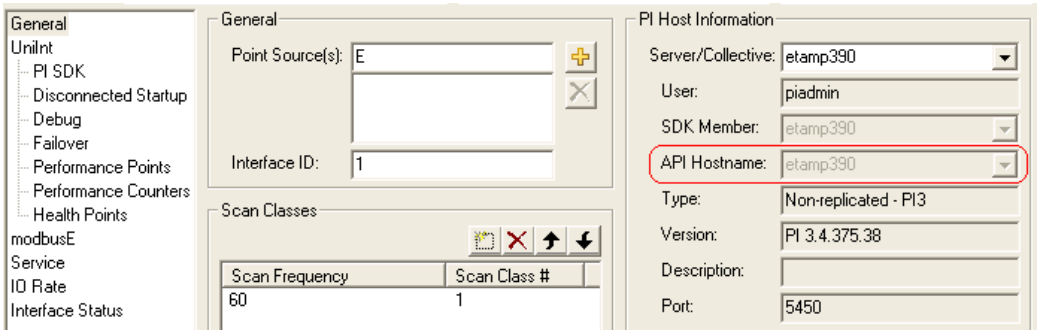
You can override this option by not checking the *Replicate data to all PI Data collective member nodes* check box. Then, uncheck (or check) the PI Data collective members as desired.



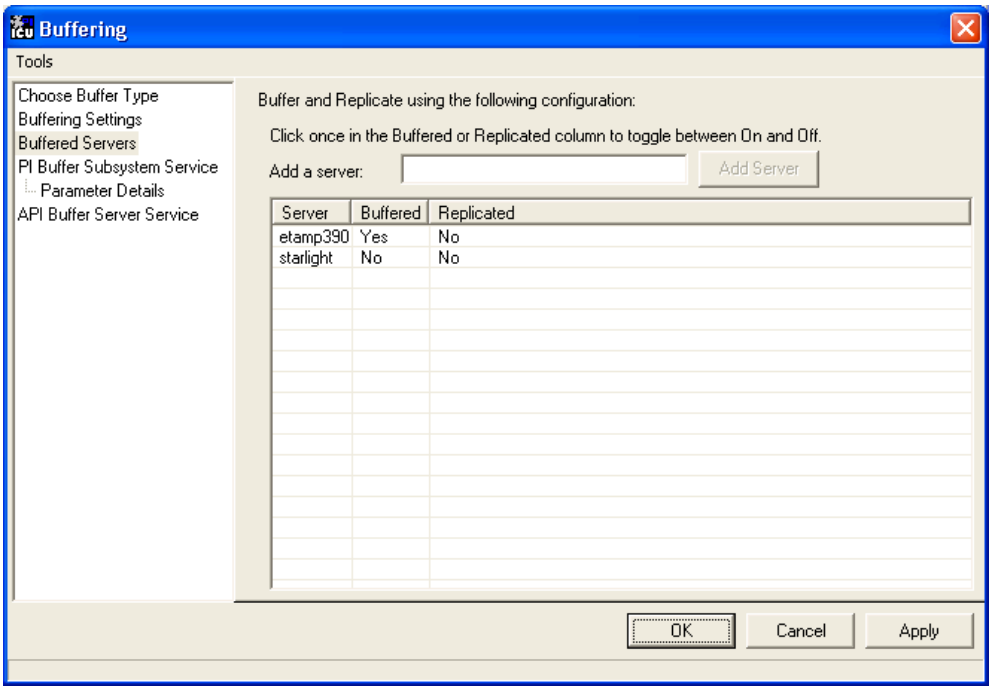
Bufserv

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

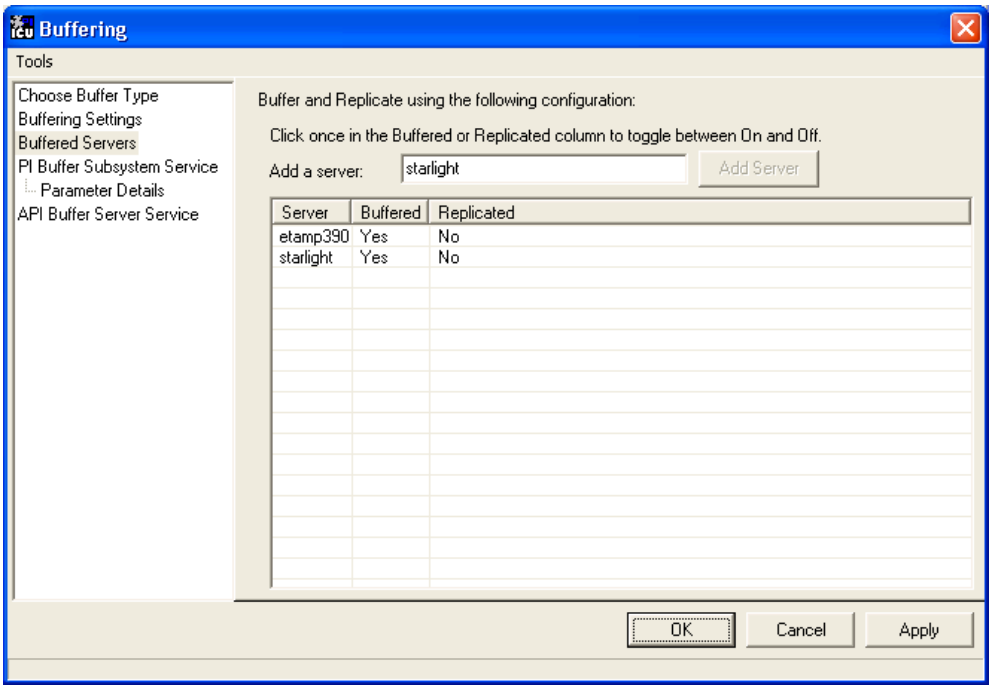
If the PI Data Archive to which you want Bufserv to buffer data is not in the Server list, enter its name in the *Add a server* box and click the *Add Server* button. This PI Data Archive name must be identical to the **API Hostname** entry:



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



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



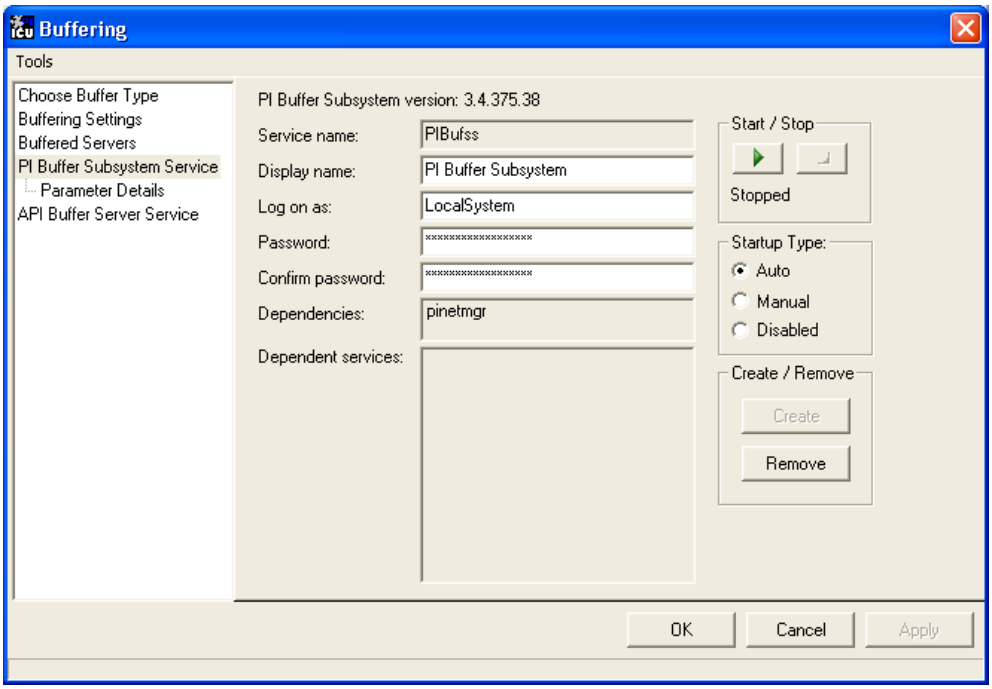
Installing Buffering as a Service

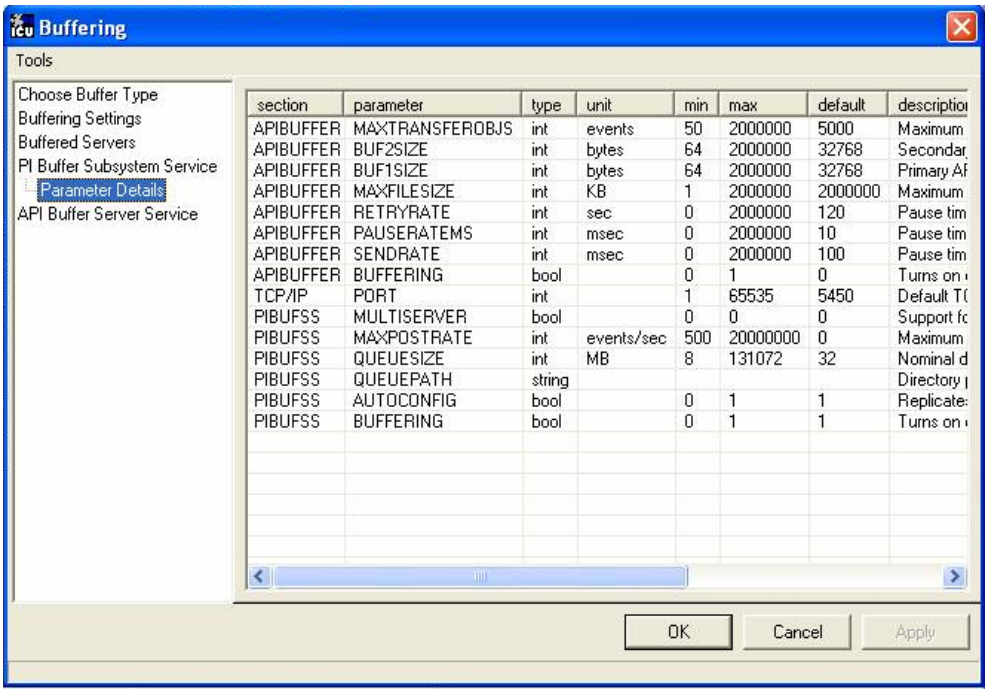
Both the PIBufss and Bufserv applications run as a Service.

PI Buffer Subsystem Service

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

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

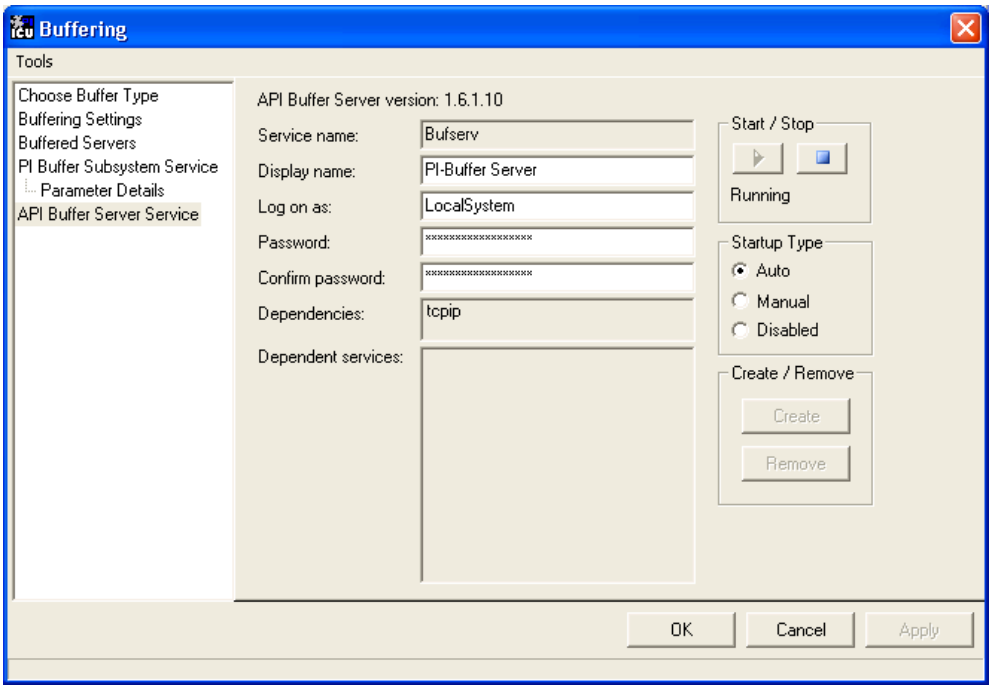




API Buffer Server Service

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

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

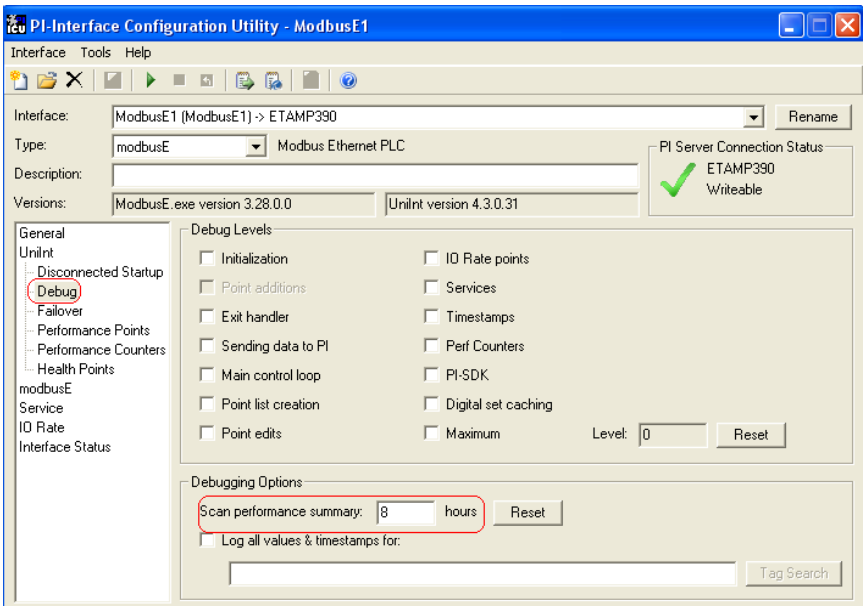

Buffering
Tools
Choose Buffer Type
Buffering Settings
Buffered Servers
PI Buffer Subsystem Service
Parameter Details
API Buffer Server Service
API Buffer Server version: 1.6.1.10
Service name:
Bufserv
Display name:
PI-Buffer Server
Log on as:
LocalSystem
Password:
Confirm password:
Dependencies:
tcpip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

Chapter 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**.

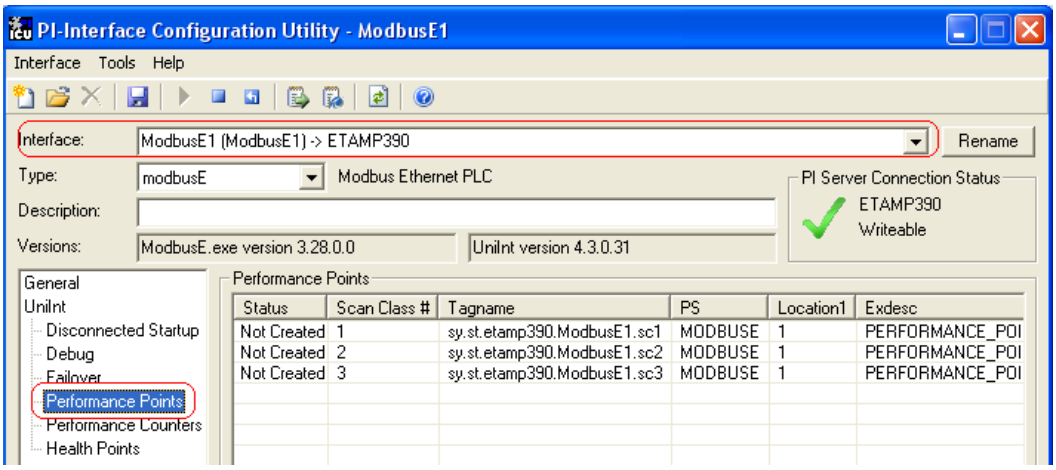
Some of the points that follow refer to a "performance summary interval". This interval is 8 hours by default. You can change this parameter via the *Scan performance summary* box in the *UniInt – Debug* parameter category page:



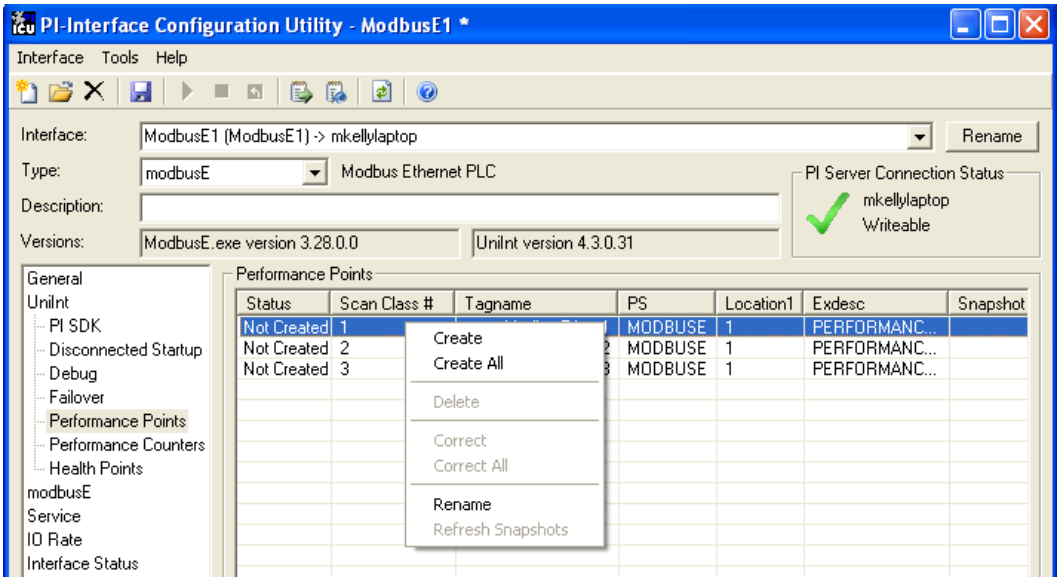
Scan Class Performance Points

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

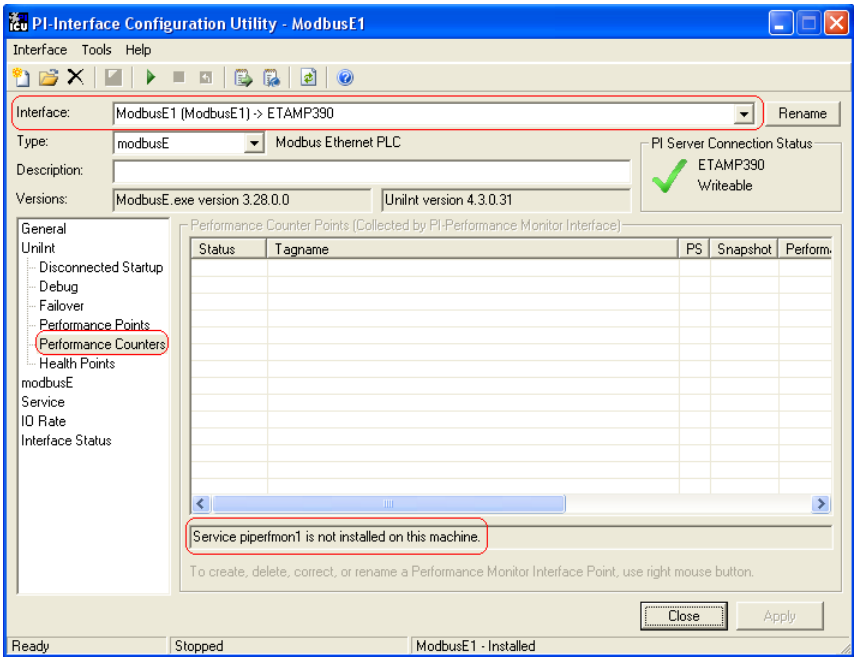
When running as a Service or interactively, this interface exposes performance data via Windows Performance Counters. Such data include items like:

- 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

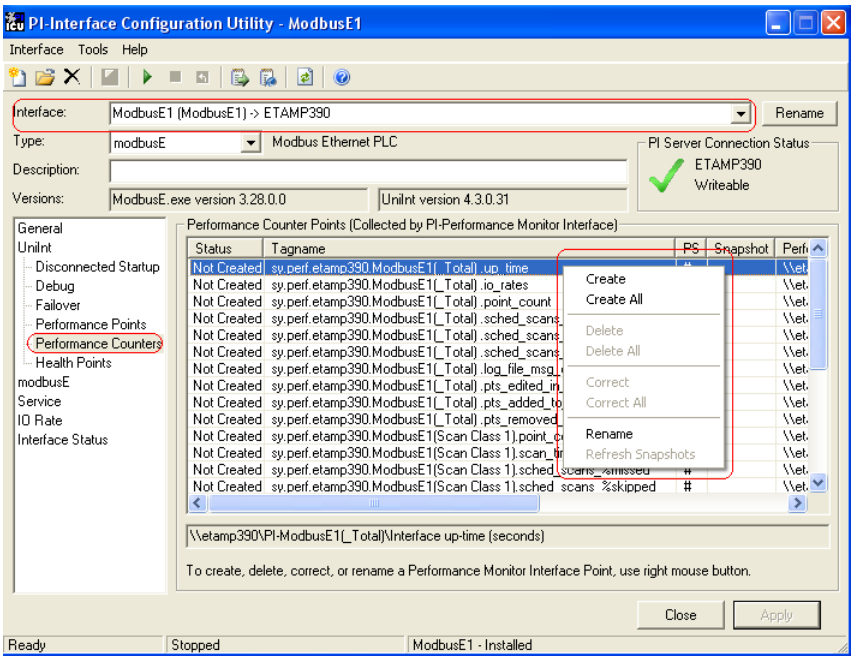
There are two types or instances of Performance Counters that can be collected and stored in PI Points. The first is (_Total) which is a total for the Performance Counter since the interface instance was started. The other is for individual scan classes (Scan Class *x*) where *x* is a particular scan 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 Data Archive the interface is sending its data to, you cannot use the ICU to create any interface instance's Performance Counters Points:



After installing the PI Performance Monitor Interface as a service, select this interface instance from the *Interface* drop-down list, then click *Performance Counters* in the parameter categories pane, and right-click on the row containing the Performance Counters Point you wish to create. This will 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 Data Archive. If the interface is properly communicating with the PI Data Archive then the value of the counter is '0'. If the communication to the PI Data Archive goes down for any reason then the value of the counter will be '1'. Once the interface is properly communicating with the PI Data Archive again then the value will change back to '0'.

"Points added to the interface" (.pts_added_to_interface)

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

"Points edited in the interface"(.pts_edited_in_interface)

The *.pts_edited_in_interface* Performance Counters Point indicates the number of point edits the interface has detected. The interface detects edits for those points whose PointSource attribute matches the **`/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)

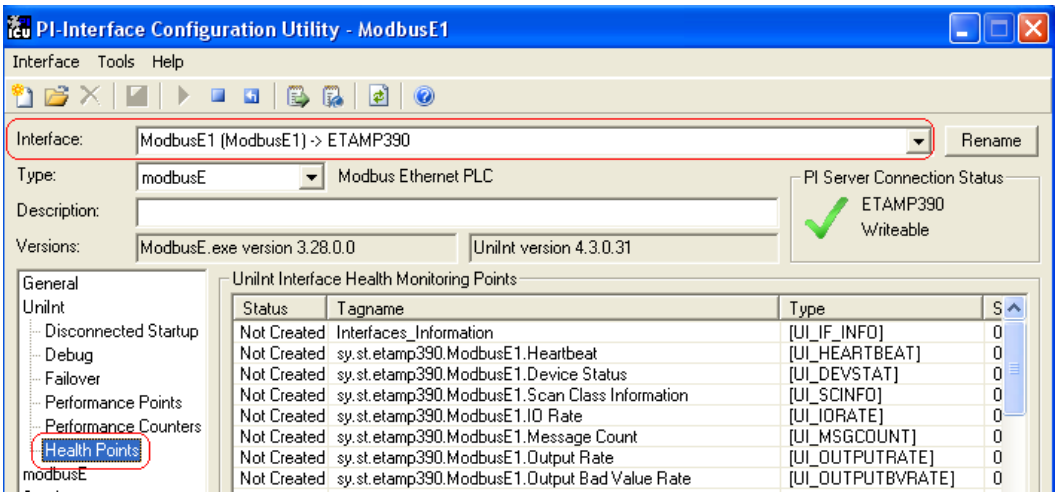
A *.scan_time* Performance Counter Point is available for each scan class of this interface.

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.

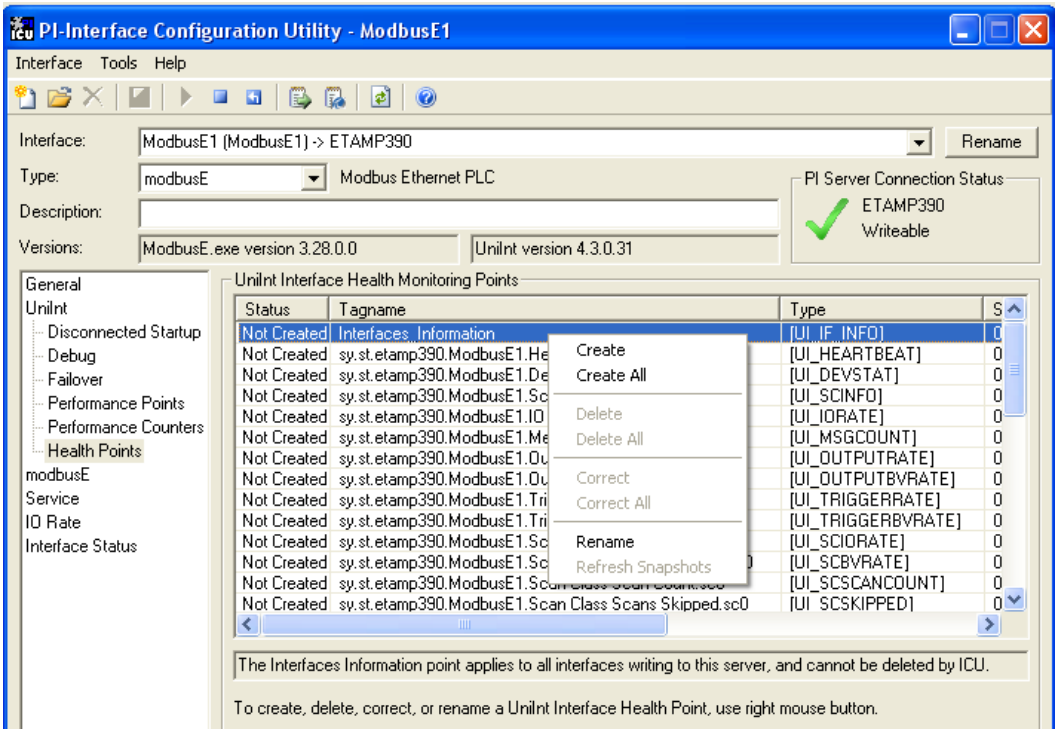
The ICU uses a naming convention such that the tag containing "(Scan Class 1)" (for example, "`sy.perf.etamp390.E1(Scan Class 1).scan_time`") refers to scan class 1, "(Scan Class 2)" refers to scan class 2, and so on.

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]

For a Health Tag with an Extended Descriptor attribute that contains [UI_DEVSTAT], the interface writes the following values:

- "1 | Could not read web page." – If the interface cannot connect to the web site this message is written to the Health tag.

Refer to the `UniInt Interface User Manual.doc` file for more information about how to configure Health Tags.

[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 Data Archive.

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

[UI_SCINDEVSCANTIME]

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

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

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

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

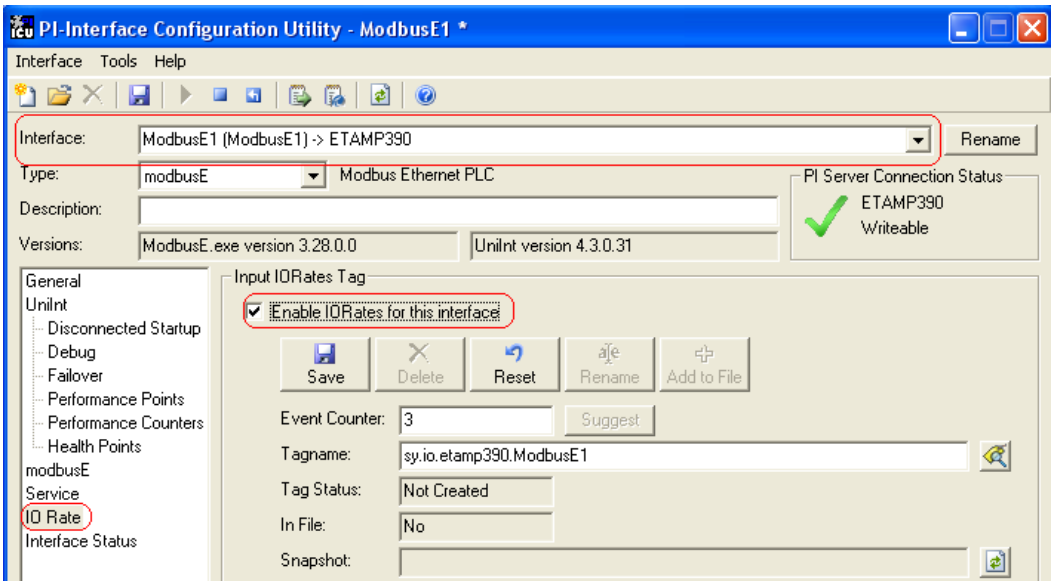
The interface updates the value of this point at the completion of the associated scan.

I/O Rate Point

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

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

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



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

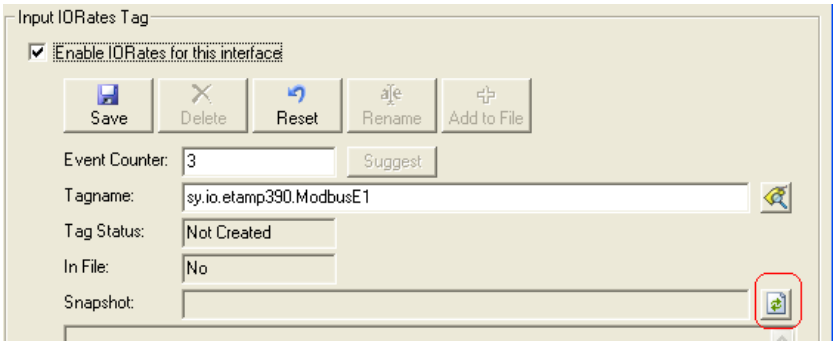
You need to restart the interface in order for it to write a value to the newly created I/O Rate point. Restart the interface by clicking the *Restart* button:

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

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

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

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



Enable IORates for this Interface

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

Event Counter

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

Tagname

The tag name listed 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 Data Archive

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 Data Archive for a previously defined I/O Rate tag.

Interface Status Point

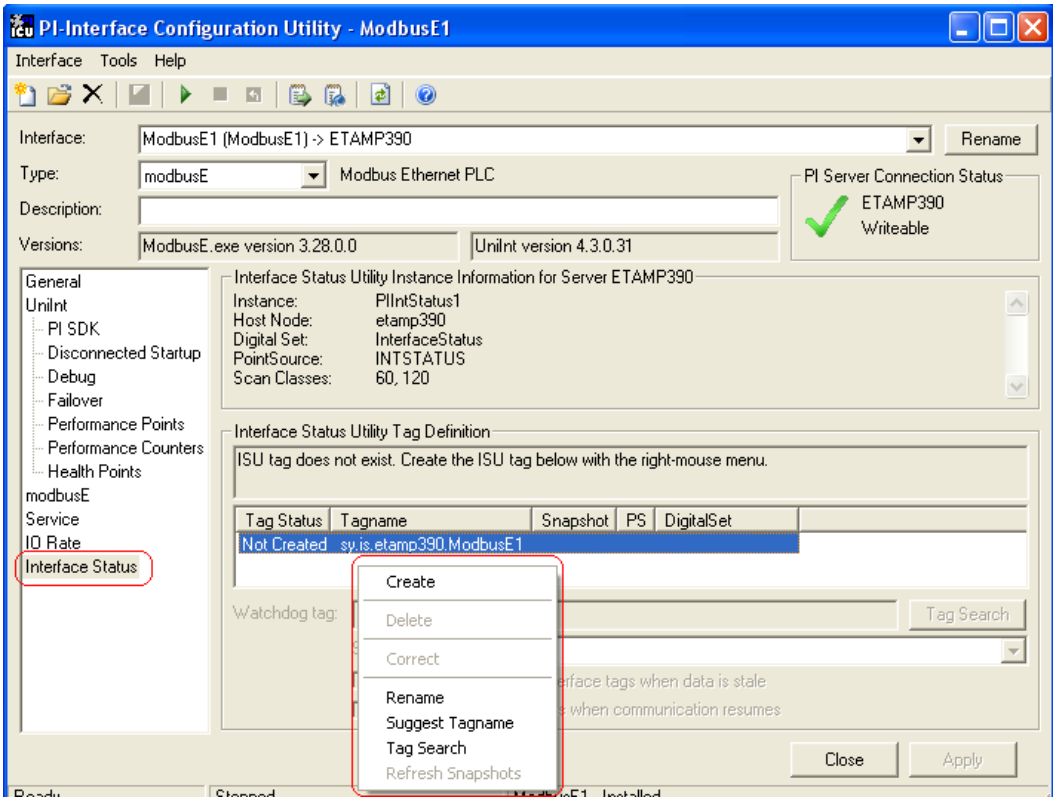
The PI Interface Status Utility (ISU) alerts you when an interface is not currently writing data to the PI Data Archive. This situation commonly occurs if

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

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

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

If you have used the ICU to configure the PI Interface Status Utility Interface on the PI Data Archive Node, the ICU allows you to create the appropriate ISU point. Select this interface from the *Interface* 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.

Troubleshooting Differences Between the ICU Setup and the Interface

For some web pages during marker setup, the ICU will show one thing in the preview tab and another thing in the Validate Markers window. This is due to differences in how the ICU gets and parses web pages and how the interface gets and parses pages. In order to maintain ease of configuration, the ICU and the interface use slightly different methods to get and parse their web pages.

Check the Proxy and HTTP Authentication Settings

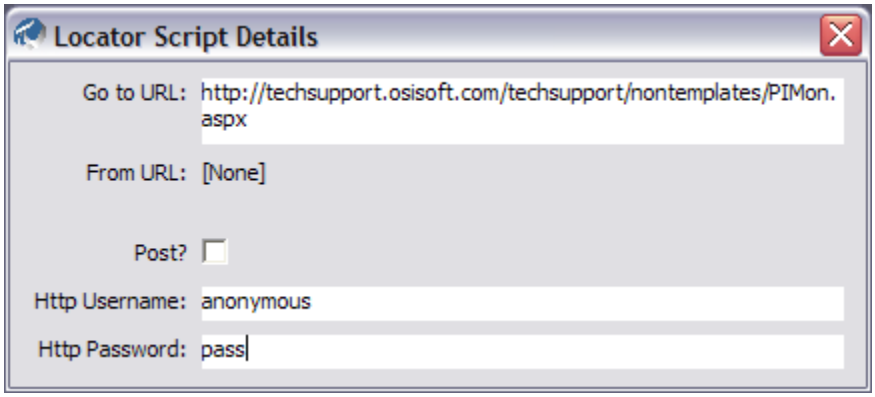
When configuring the HTML interface using the ICU, sometimes you will come across pages where an authentication screen will appear. The ICU cannot automatically record this. Make sure that if a proxy server, username, and password are required, that they are specified in the first box that appears after you click **Record New**.

Also, make sure that if a web site requires authentication for a page, that those credentials are entered in the Locator Script Details window described in Steps for Creating a New HTML Locator Script of section html Interface Page.

Connecting to an FTP

A webpage that is being stored on an FTP site can be configured for anonymous login access only. The interface does not currently support connections to an FTP which requires authentication. There is an additional step required in setting up a connection to an FTP site with anonymous login. Record a new locator script. After the Record Locator Script page is successfully loaded, click **Path to Current Location** in the box on the upper right corner of the page. Although the page is allowing anonymous access, the interface still requires that a login be entered in the HTTP Username and Password section.

After entering the information:



Close the **Locator Script Details** dialog box, and then click **OK** on the **Record Locator Script**, and save the changes.

View the HTML Source Externally

Sometimes the HTML source code that is downloaded when editing the markers may be different from the source code downloaded using the Curl library, which is what is used when validating the markers and when the interface itself is running. Click **Validate Markers**, and then click **See HTML** to see the HTML source code. Save that source code to a file, and then open that file in Internet Explorer. This is what the interface will see when it is parsing the page. There is no single answer as to why the HTML may be different.

Look For JavaScript Include Directives

Because of the major change in how the interface downloads and parses HTML pages, JavaScript include directives do not work correctly. Many times, JavaScript code that is common to multiple web pages is placed in a single file and referenced by multiple web pages. Versions of the HTML interface prior to 2.0 were able to fetch these JavaScript include files automatically, just as images on a page are fetched automatically when a page is loaded in Internet Explorer. Since version 2.0, however, the interface is not able to fetch these pages automatically. Some critical JavaScript code may be missing from the target page and the data on the page may not show up properly.

To see if this is a problem, view the source code for the page by clicking **Validate Markers**. If there is a section on the page that includes the following code, it is trying to include an external JavaScript source file:

```
<SCRIPT LANGUAGE="JavaScript" SRC="somefile.js"></SCRIPT>
```

The important point is that there is a `SRC="somefile.js"` in the declaration.

Message Logs

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

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

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

Messages

Message	`No interface configuration file specified, exiting`
Meaning	There was no XML configuration file specified (using `/htmlconfigfile`) in the interface startup file.

Message	`Warning: No interface ID was specified, all points with pointsource X will be used`
Meaning	When a non-numeric interface ID or no interface ID is specified, all points with the corresponding pointsource will be treated as belonging to this interface.

Message	`Tag X (D) has an invalid instrumenttag (), point rejected.`
Meaning	There must be an instrumenttag specified for all points, and the instrumenttag must either be in the list of data markers or must be a semi-colon delimited list of data markers.

Message	`Tag X (D) has a data marker (someinstrumenttag) that does not exist in the XML configuration doc, point rejected.`
Meaning	The instrumenttag must either be in the list of data markers or must be a semi-colon delimited list of data markers.

Message	`Tag X (D) has multiple data markers defined, so digital state errors will not be reported for this tag.`
Meaning	For points with multiple data markers defined, it is assumed that there will be multiple timestamps for the values. Therefore it is impossible to determine what timestamp to use when sending a digital state error to PI.

Message	`HTML parsing error (parser errors will not be logged until a successful parse): Some error.`
Meaning	There was a problem parsing the HTML page. Errors are logged only once until the parser is working again.

Message	`Error: Timestamp|Data marker <marker> for tag <tag> could not be read from page, errors will not be repeated for this tag until the tag is read successfully.`
Meaning	The data or timestamp marker was not found on the page. Check the marker definitions using the PI ICU to make sure the page has not changed.

Message	`Error: Error executing search and replace for timestamp|data marker <marker> for tag <tag>, errors will not be repeated for this tag until the tag is read successfully.`
Meaning	There some kind of problem using the RegExp engine. Check the search and replace settings in the marker definitions.

Message	`'value' could not be converted to <data type> for timestamp|data marker <marker> for tag <tag>, errors will not be repeated for this tag until the tag is read successfully.`
Meaning	There was an error converting a value read from the page to the desired data type. For timestamp markers, this is the date data type. For data markers, this is the pointtype of the target point.

Message	`Downloading HTML from <URL> timed out, scan skipped.`
Meaning	The download timeout has been passed when downloading the HTML page from its source.

Message	`Error navigating to <URL> on attempt D (<Error>), trying again|no more retries.`
Meaning	Navigating to the page failed. The interface will try to navigate to a page up to 3 times before finally concluding that it has failed. Check the proxy settings, http authentication settings, and the URL.

Message	`At least one point for Pointsource H was found where Location1 does not match with /ID=x.`
Meaning	This is not an error. It just means that there are points that match this interface's pointsource, but the location1 does not match the id, meaning that there are several copies of this interface that need to run at the same time.

System Errors and PI Errors

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

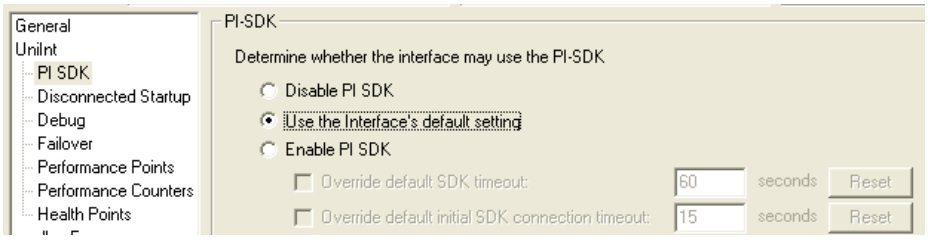
Error Descriptions

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

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

Appendix B. PI SDK Options

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



Disable PI SDK

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

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

Use the Interface's default setting

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

Enable PI SDK

Select *Enable PI SDK* to tell the interface to use the PI SDK. Choose this option if the PI Data Archive version is earlier than 3.4.370.x or the PI API is earlier than 1.6.0.2, and you want to use extended lengths for the Tag, Descriptor, ExDesc, InstrumentTag, or PointSource point attributes. The maximum lengths for these attributes are:

Attribute	Enable the Interface to use the PI SDK	PI Data Archive earlier than 3.4.370.x or PI API earlier than 1.6.0.2, without the use of the PI SDK
Tag	1023	255
Descriptor	1023	26
ExDesc	1023	80
InstrumentTag	1023	32
PointSource	1023	1

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

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

Appendix C. Plug-in Architecture

The PI Interface for HTML supports COM plug-ins in order to customize its functionality. There are four main customizable actions that can be taken by plug-ins. They are:

- Dynamic URL generation
- Timestamp generation
- Value generation
- HTML modification

COM is used as the mechanism to activate plug-ins. This makes it very simple to use Microsoft Visual Basic to create plug-ins. It's as simple as adding "implements PIHTMLPlugin" to the top of a project's code page. There is also a Visual Basic sample in the `PIPC\interfaces\HTML\Plugins\Samples\` directories.

Dynamic URL Generation

Dynamic URL generation is useful when there is a page you are trying to read on a regular basis, whose URL changes every so often. For example, if there is a page that has today's weather, and the date is part of the URL, a dynamic URL will need to be generated. So, if the URL for that page looks like:
`http://www.yourfavoriteweathersite.com/OAK_08_12_2002.html`

this will obviously be different for each day the weather needs to be read from the site.

With dynamic URL generation, the plug-in is given a "dummy" URL that the user specifies in the HTML locator script, which is configured using the PI ICU or the simpler PI Interface for HTML configuration tool. Continuing the weather example, this URL could be: `http://www.yourfavoriteweathersite.com/OAK_[month]_[day]_[year].html`. The plug-in would then be responsible for making any text substitutions in this URL. So the plug-in could look for [month] and replace it with the current month, and so on.

Timestamp and Value Generation

Timestamp and value generation are two separate features, but are almost identical, so they will be covered simultaneously.

Many times, the timestamps on the HTML page are not exactly what you want to be sending to PI. For example, there may be a site that lists some alternate representation for hours. Instead of showing 12:00 am, 1:00 am, 2:00 am, etc., the site may have a table with a column heading "hour", and the column will list 0 (for 12:00 am), 1 (for 1:00 am), etc. The plug-in would receive this old value and do the appropriate math on this and return an actual timestamp to the interface.

There may also be sites where the values themselves are not exactly how you want to send them to PI. There may need to be some mathematical transformation performed on the data. For example, there may be raw data on a web site from some system that is meant to be taken

as the exponent for the exponential function (ex). The plug-in would receive this raw value, perform the transformation on the value, and send the new value back to the interface.

HTML Modification

HTML Modification is a new feature added to version 2.0 of the HTML interface. This feature allows the HTML downloaded from the web page to be modified before being sent to the parser. This feature is only accessible through the IPIHTMLPlugins2 COM interface.

There are several reasons someone might want to modify the HTML before the interface parses the page. The HTML might be malformed, and thus might need to be modified in order for it to be parsed. For example, a page may use the wrong order to close HTML tags, like in the following malformed snippet:
`<FONT color="blue"><B>Text in here</FONT></B>.`
HTML requires that tags be closed in the reverse order that they were opened. So a plug-in might be coded to search for this particular section of the page and re-write it this way:
`<FONT color="blue"><B>Text in here</B></FONT>.`

Another reason to modify the page is to deal with text files. Some web pages are plain text files with no HTML markup at all. To tell if a page is just plain text, open the page in a web browser and view its source. If there is no HTML markup in the page, it is plain text. Putting `<PRE>` before the text and `</PRE>` after the text makes it a little easier to use regular expressions to search the text, because the parser replaces all returns, tabs, and multiple spaces with a single space when parsing if the text is not enclosed in `<PRE>` tags.

Receiving Pre-Transformed Information from the Interface

As stated above, the plug-in receives pre-transformed information from the interface. This is done using the interface's HTML Locator script functionality, for dynamic URL generation, and the interface's timestamp and data marker functionality, for timestamp and value generation. So, if you want the pre-transformed URL to look like `http://www.yourfavoriteweathersite.com/OAK_[month]_[day]_[year].html`, you need to set that as the URL when you are configuring the locator script. If this is set as the target URL before a plug-in is selected in the configuration utility, the interface will try to actually navigate to this page as it is written above. Of course, this page will likely not exist. So be sure to have the plug-in selected in the **Misc** dialog box before testing.

The timestamp and data markers are used to determine the pre-transformed timestamp and value information. Whatever is on the HTML page at the locations pointed to by the timestamp and data markers is what the plug-in will receive. The plug-in is then responsible for performing the transformation, and returning a modified timestamp or value.

The COM Interfaces

The following is the IDL for the COM interface used as the bridge between the interface and any plug-in.

```
interface IPIHTMLPlugin : Idispatch
{
          [id(1), helpstring(„method SetDocument“)] HRESULT
SetDocument([in] IHTMLDocument2 * newVal);
```

```
          [id(2), helpstring(„method GetURL“)] HRESULT GetURL([in] BSTR
BSTROld, [in,out] BSTR * BSTRURL, [in,out] VARIANT_BOOL * vbUsingPost);
          [id(3), helpstring(„method ProcessTimestamp“)] HRESULT
ProcessTimestamp([in] BSTR BSTRTimestampMarker, [in] BSTR
BSTROldTimestamp, [out, retval] VARIANT * varNewTimestamp);
          [id(4), helpstring(„method ProcessData“)] HRESULT
ProcessData([in] BSTR BSTRDataMarker, [in] BSTR BSTROldData, [out, retval]
VARIANT * varNewData);
          [id(5), helpstring(„method ReleaseDocument“)] HRESULT
ReleaseDocument();
};
interface IPIHTMLPlugin2 : Idispatch
{
   [id(1), helpstring(„method ProcessDownloadedHTML“)] HRESULT
ProcessDownloadedHTML([in] BSTR BSTROldHTML, [in, out] BSTR *
BSTRNewHTML);
};
```

The following is a skeleton of what the interface methods would look like when implemented in VB.

```
Implements IPIHTMLPlugin
Implements IPIHTMLPlugin2 'Optional
Private Sub IPIHTMLPlugin_SetDocument(ByVal newVal As MSHTML.IHTMLDocument2)
End Sub
Private Sub IPIHTMLPlugin_GetURL(ByVal BSTROld As String, BSTRURL As String,
vbUsingPost As Boolean)
End Sub
Private Function IPIHTMLPlugin_ProcessTimestamp(ByVal BSTRTimestampMarker As
String, ByVal BSTROldTimestamp As String) As Variant
End Function
Private Function IPIHTMLPlugin_ProcessData(ByVal BSTRDataMarker As String,
ByVal BSTROldData As String) As Variant
End Function
Private Sub IPIHTMLPlugin_ReleaseDocument()
End Sub
'Only required if Implements IPIHTMLPlugins2 is used
Private Sub IPIHTMLPlugin2_ProcessDownloadedHTML(ByVal BSTROldHTML As
String, BSTRNewHTML As String)
End Sub
```

There are five required and one optional function that need to be implemented by a plug-in. The required ones are `SetDocument`, `GetURL`, `ProcessTimestamp`, `ProcessData`, and `ReleaseDocument`. The optional one is `ProcessDownloadedHTML`.

Implementing the IPIHTMLPlugin COM interface is required for a plug-in, even if none of its functionality is required. Implementing the IPIHTMLPlugin2 COM interface is optional.

SetDocument, ReleaseDocument

`SetDocument` and `ReleaseDocument` are currently not called by the interface. They are included in the COM interface as a future enhancement in case a future plug-in developer decides that he needs to store a reference to the IHTMLDocument2 object used by the interface.

GetURL

`GetURL` is called after the locator script is read but before the navigation to the URL is handled. `BSTROld` is the original URL stored in the locator script. This may be used by the

plug-in developer, or it may be ignored. BSTRURL is the buffer for the new URL. This should be set by the plug-in developer before returning from GetURL. Even if there is no change desired, BSTRURL should at least be set to mirror the original URL, which is passed in BSTROld. So at a minimum, this function should contain logic that sets the value of BSTRURL to BSTROld. If there are any query parameters for a POST or a GET query, they should be appended to the end of the URL as if it were a GET query (even if it is a POST query). VbUsingPost should be set to True if the request is meant to be a POST request. Otherwise, it should be set to False for a GET query.

ProcessTimestamp

ProcessTimestamp is called after a timestamp marker has been read off the HTML page by the interface but before the timezone offset is applied and before the timestamp/value pair is sent to PI. The original contents of the timestamp marker are passed to this interface method, and it is up to the plug-in developer to transform and return the new timestamp. BSTRTimestampMarker is the name of the timestamp marker being sent to the plug-in. This is useful for identifying which timestamp is being currently processed, if there is more than one. BSTROldTimestamp is the timestamp as read off the HTML page. The return value is set as the transformed timestamp.

ProcessData

ProcessData is called after a data marker has been read off the HTML page by the interface but before the timestamp/value pair is sent to PI. The original contents of the data marker are passed to this interface method, and it is up to the plug-in developer to transform and return the new value. BSTRDataMarker is the name of the data marker being sent to the plug-in. This is useful for identifying which piece of data is being currently processed, if there is more than one. BSTROldData is the value as read off the HTML page. The return value is set as the transformed value.

ProcessDownloadedHTML

ProcessDownloadedHTML is called after the page has been downloaded but before it is parsed. This gives the user a chance to change the downloaded HTML for whatever reason.

Plug-in Registration and Categorization

A plug-in needs to be registered and categorized before it can be used by the PI Interface for HTML. Registration is the process by which any COM server (in this case, a plug-in) is registered with Windows so it can be called by an application (in this case, the PI Interface for HTML). Categorization is the process by which a COM server (plug-in) is registered as belonging to a certain category. Categorization is normally not required for COM servers, but for the PI Interface for HTML, it is required. This is so the configuration utility can more easily find all plug-ins that are actually valid PI Interface for HTML plug-ins.

COM server (plug-in) registration is done by starting up a Command Prompt from Windows. The command to register a COM server (plug-in) DLL is regsvr32 *<path to plug-in>*. The command to unregister a COM server (plug-in) DLL is regsvr32 -u *<path to plug-in>*. However, this step can be ignored if you use the configuration utility (either the

PI ICU or the simpler configuration utility provided with the interface) to browse for the plug-in.

Quick Registration and Categorization

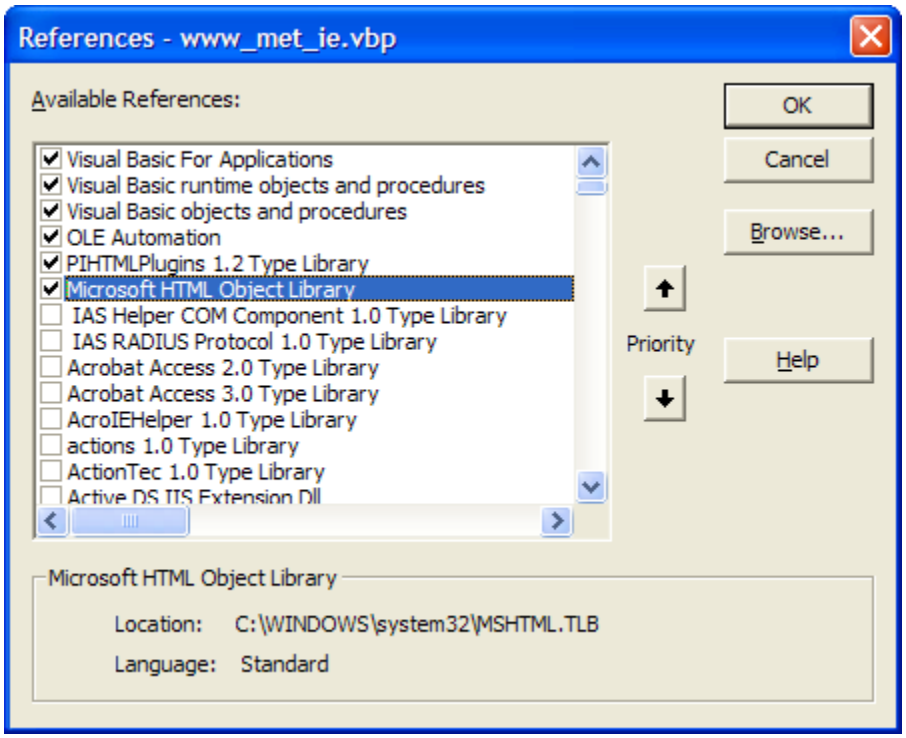
Because Visual Basic does not allow access to `DllRegisterServer`, the configuration utility can register and categorize any plug-ins. After installing the plug-in anywhere on the system, open the configuration utility, click **Misc**, and find the plug-in section in the dialog box. Click the **Browse** button and browse for the plug-in DLL. After selecting it and clicking **OK**, it will be registered and categorized.

If a plug-in is not registered and categorized, it cannot be used by the PI Interface for HTML.

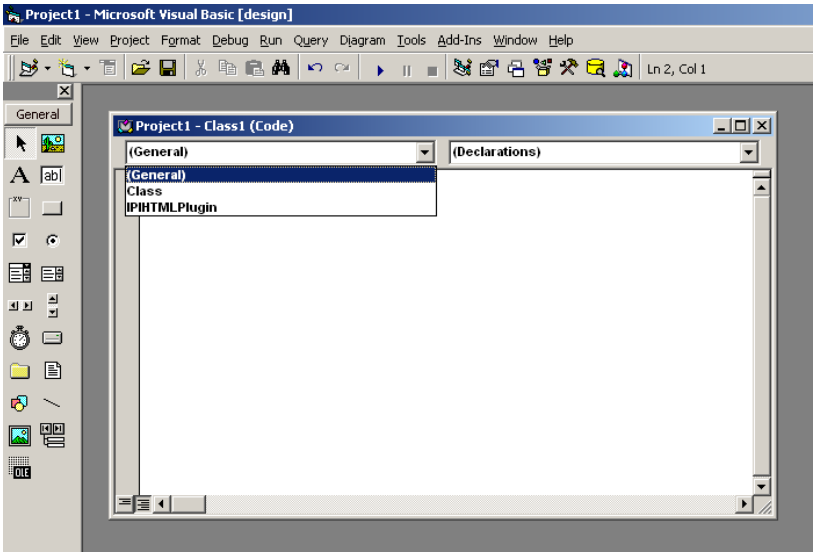
Creating a Visual Basic Plug-in

Creating a plug-in is extremely simple using Visual Basic. To create a plug-in, start Visual Basic. Create a new ActiveX DLL.

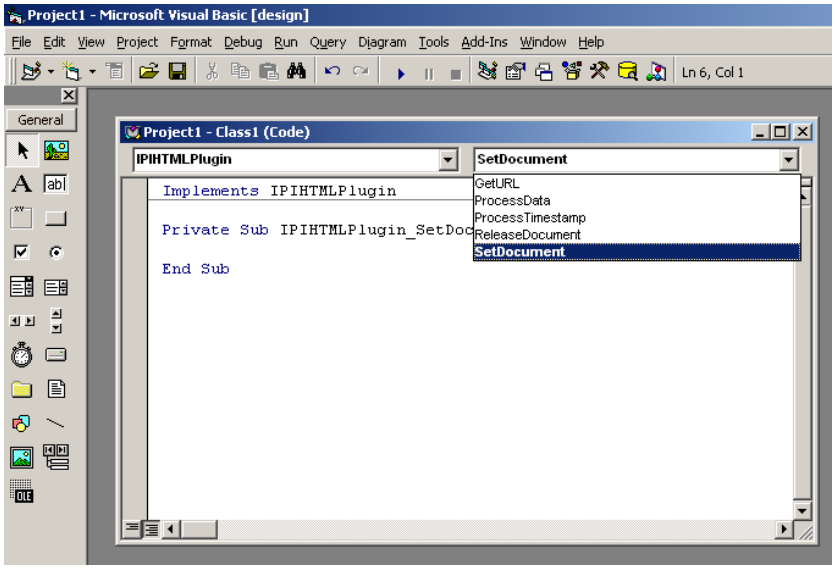
1. On the **Project/References** menu option, click the checkbox next to **PIHTMLPlugins** 1.2 Type Library and **Microsoft HTML Object Library**.



2. Click **OK**. Then add the line `Implements IPIHTMLPlugin` to the top of your code, and you're ready to start filling in the interface methods.
3. Select the **IPIHTMLPlugin** item from the left drop-down menu.



4. Select each of the items in the right drop-down menu, until all `IPIHTMLPlugin` methods have been added to the code page.



At a minimum, `IPIHTMLPlugin_GetURL`, `IPIHTMLPlugin_ProcessData`, and `IPIHTMLPlugin_ProcessTimestamp` should contain this code:

```
Private Sub IPIHTMLPlugin_GetURL(ByVal BSTROld As String, BSTRURL As String,
vbUsingPost As Boolean)
    BSTRURL = BSTROld
End Sub
Private Function IPIHTMLPlugin_ProcessData(ByVal BSTRDataMarker As String,
ByVal BSTROldData As String) As Variant
    IPIHTMLPlugin_ProcessData = BSTROldData
End Function
Private Function IPIHTMLPlugin_ProcessTimestamp(ByVal BSTRTimestampMarker As
String, ByVal BSTROldTimestamp As String) As Variant
    IPIHTMLPlugin_ProcessTimestamp = BSTROldTimestamp
End Function
Private Sub IPIHTMLPlugin_ReleaseDocument()
End Sub
Private Sub IPIHTMLPlugin_SetDocument(ByVal newVal As MSHTML.IHTMLDocument2)
End Sub
To also implement the ProcessDownloadedHTML routine, add "Implements
IPIHTMLPlugin2" to the top of the code page, and add the subroutine shown
below:
Private Sub IPIHTMLPlugin2_ProcessDownloadedHTML(ByVal BSTROldHTML As
String, BSTRNewHTML As String)
    BSTRNewHTML = BSTROldHTML
End Sub
```

5. Make the DLL using the **File** menu option **Make <pluginname>.dll**, and the dll is ready to be registered and categorized with the configuration utility.

Appendix D. Terminology

To understand this interface manual, you should be familiar with the terminology used in this document.

Buffering

Buffering refers to an interface node's ability to store temporarily the data that interfaces collect and to forward these data to the appropriate PI Data Archives.

N-Way Buffering

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

ICU

ICU refers to the PI Interface Configuration Utility. The ICU is the primary application that you use to configure PI interface programs. You must install the ICU on the same computer on which an interface runs. A single copy of the ICU manages all of the interfaces on a particular computer.

You can configure an interface by editing a startup command file. However, OSIsoft discourages this approach. Instead, OSIsoft strongly recommends that you use the ICU for interface management tasks.

ICU Control

An ICU Control is a plug-in to the ICU. Whereas the ICU handles functionality common to all interfaces, an ICU Control implements interface-specific behavior. Most PI interfaces have an associated ICU Control.

Interface Node

An interface node is a computer on which

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

PI API

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

PI Data collective

A PI Data collective is two or more replicated PI Data Archives that collect data concurrently. PI Data collectives are part of the High Availability environment. When the primary PI Data Archive in a PI Data collective becomes unavailable, a secondary PI Data

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 Data Archive. Some PI interfaces, in addition to using the PI API, require the use of the PI SDK.

PI Data Archive Node

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

PI SMT

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

Pipc.log

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

Point

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

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

Service

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

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

Tag (Input Tag and Output Tag)

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

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

Appendix E. 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 F. Revision History

Date	Author	Comments
03-May-2001	LNG	Restarted manual using Skeleton version 1.08
23-May-2001	CG	Skeleton 1.09; removed Program Files from directory paths; added a more complete sample .bat file; fixed headers & footers; fixed page numbering
01-Nov-2001	LNG	Updated manual for 1.0.3 release.
04-Feb-2002	LNG	Updated for 1.0.5 release.
10-Jul-2002	LNG	Updated for 1.1.0 release.
29-Sep-2004	LNG	Updated for 1.1.3 release. Added XP DCOM config, and added more interface options. Added Appendix C.
22-Oct-2004	Mkelly	Fixed headers & footers. Added section on Configuring Buffering with PI ICU. Made manual FINAL.
25-Feb-2005	LNG	Updated for 1.2.0.0 release. Added section about converting XML configuration file. Added section about the new CURL library used to download pages. Updated screenshots for the new PI ICU control.
29-Mar-2005	LNG	Updated for 1.2.0.4 release. Added ProcessDownloadedHTML section. Added note about allowable timestamp formats to the introduction.
2-May-2005	Mkelly	Fixed installation directory references. Included missing support feature items from latest skeleton manual. Fixed headers/footers and TOC. Accepted all changes and made Final.
12-Jul-2005	LNG	Updated for version 2.0 release. Added section about the Validate Markers button. Added troubleshooting for differences between ICU and interface operation.
22-Apr-2008	LNG	Updated for version 2.2.0.63 release. Added location2 description. Added file:// URL format requirement. Updated system requirements. Removed /maxiescans option.
22-Apr-2008	BJM	Using Interface Manual Skeleton 2.5.2. Including section on FTP connections. Including revised section on example and setup instructions.
10-Sep-2008	Mkelly	Version 2.2.0.63, Revision A; Updated all cross references to hyperlinks, removed all references to UniInt End User Document and replaced with UniInt Interface User Manual, remove all “PI-‘ and replaced with just PI and a space. Fixed size of screenshots. Removed all NT4 and UNIX references. Fixed headers and footers.
08-Oct-2008	Mkelly	Version 2.2.0.63, Revision B; Updated to skeleton 3.0.4, fixed all hyperlinks and references.
03-Sep-2012	Sbranscomb	Version 2.2.0.63 Revision C: Updated to Skeleton

Date	Author	Comments
		Version 3.0.35
19-Feb-2013	Mkelly	Version 2.2.0.63 Revision D: Update to Skeleton Version 3.0.36
26-Feb-2013 16-Dec-2013	OPopivshchyi LDaley MHruzik	Version 2.3.0.x: Addressed interface name change; Updated references to .Net Framework v2.0 to v4.0; Added statements regarding passwords encryption; Added Phase 2 failover sections; Removed reference to obsolete plugin projects samples. Added new security content. Update to Skeleton Version 3.0.38.
02-Apr-2014	MHruzik	Updated terms for PI Data Archive and PI Data collective
02-Apr-2014	ZRyska	Finished name change and small corrections