



AVEVA™ InTouch HMI  
formerly Wonderware

Data Management Guide

© 2021 AVEVA Group plc and its subsidiaries. All rights reserved.

No part of this documentation shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of AVEVA. No liability is assumed with respect to the use of the information contained herein.

Although precaution has been taken in the preparation of this documentation, AVEVA assumes no responsibility for errors or omissions. The information in this documentation is subject to change without notice and does not represent a commitment on the part of AVEVA. The software described in this documentation is furnished under a license agreement. This software may be used or copied only in accordance with the terms of such license agreement.

ArchestrA, Aquis, Avantis, Citect, DYNsIM, eDNA, EYESIM, InBatch, InduSoft, InStep, IntelTrac, InTouch, OASyS, PIPEPHASE, PRISM, PRO/II, PROVISION, ROMeo, SIM4ME, SimCentral, SimSci, Skelta, SmartGlance, Spiral Software, Termis, WindowMaker, WindowViewer, and Wonderware are trademarks of AVEVA and/or its subsidiaries. An extensive listing of AVEVA trademarks can be found at: <https://sw.aveva.com/legal>. All other brands may be trademarks of their respective owners.

Publication date: Monday, May 3, 2021

## Contact Information

AVEVA Group plc  
High Cross  
Madingley Road  
Cambridge  
CB3 0HB. UK

<https://sw.aveva.com/>

For information on how to contact sales and customer training, see <https://sw.aveva.com/contact>.

For information on how to contact technical support, see <https://sw.aveva.com/support>.

# Contents

<b>Chapter 1 Data Management Overview .....</b>	<b>10</b>
Data Management Overview About Data Management .....	10
Working with InTouch Tags .....	11
<b>Types of InTouch Tags .....</b>	<b>11</b>
Memory Tags.....	12
Local Tags.....	13
I/O Tags .....	13
Indirect Tags .....	14
Miscellaneous Tags .....	15
Hist Trend tags.....	15
Tag ID Tags.....	15
SuperTags .....	15
Obsolete Tags .....	15
<b>Tag Properties.....</b>	<b>15</b>
Memory Tag Properties.....	16
I/O Tag Properties .....	18
<b>Remote Tag References.....</b>	<b>20</b>
<b>Chapter 2 Managing Tags with the Tagname Dictionary.....</b>	<b>21</b>
Managing Tags with the Tagname Dictionary About Managing Tags with the Tagname Dictionary .....	21
Planning Tag Usage .....	21
Creating New Tags .....	22
<b>Configuring Tag Properties .....</b>	<b>23</b>
Common Tag Properties.....	23
Tag Name Conventions.....	24
Automatically Naming Tags .....	24
Tag Comments.....	25
Understanding Tag Properties .....	25
Value Ranges, Measurement Units, and an Initial Value .....	25
Tag Deadbands .....	26
Tag Value Retention .....	27
I/O Connection.....	27
Tag Logging .....	27

Creating Discrete Tags.....	28
Creating Integer and Real Tags.....	29
Creating Message Tags.....	30
Creating I/O Tags.....	30
<b>Modifying Tags .....</b>	<b>30</b>
<b>Deleting Tags .....</b>	<b>31</b>
<b>Printing a Tag List and Usage Information .....</b>	<b>31</b>
<b>Chapter 3 System Tags.....</b>	<b>32</b>
System Tags About System Tags .....	32
System Tags Reference.....	32
<b>Chapter 4 Using Tag Dotfields to View or Change Tag Properties .....</b>	<b>38</b>
About Using Tag Dotfields to View or Change Tag Properties .....	38
Available Dotfields for Tag Types.....	38
Changing the Value Limits of a Tag .....	46
Viewing the Raw Value Limit.....	47
.MinRaw Dotfield.....	47
.MaxRaw Dotfield .....	48
Viewing the Raw Value of a Tag .....	49
.RawValue Dotfield .....	49
Viewing the Engineering Units Value Limit .....	50
.MaxEU Dotfield.....	50
.MinEU Dotfield .....	51
Changing the Engineering Units of a Tag .....	51
.EngUnits Dotfield.....	51
Viewing the Value of a Tag in Engineering Units.....	52
.Value Dotfield.....	52
Viewing or Changing Discrete Tag Messages .....	53
.OnMsg Dotfield.....	53
.OffMsg Dotfield .....	54
Viewing or Changing the Comment of a Tag.....	54
.Comment Dotfield .....	54
<b>Chapter 5 Data Access with I/O.....</b>	<b>56</b>
Data Access with I/O About Data Access with I/O .....	56
Working with OI Gateway Communication Driver .....	57
About the Gateway Communication Driver .....	57
Getting Started Workflow .....	58
Supported InTouch Communication Protocols .....	58
Dynamic Data Exchange .....	58
SuiteLink .....	58
Troubleshooting SuiteLink Communication Problems .....	59

Accessing the Server as a Standard User.....	59
OPC.....	59
OPC UA.....	60
MQTT.....	60
<b>Setting Up Access Names .....</b>	<b>61</b>
Deleting Access Names .....	62
<b>Accessing I/O Data with I/O Tags.....</b>	<b>63</b>
Configuring I/O Tag Properties.....	63
Specifying a Discrete I/O Tag.....	63
Specifying Integer and Real I/O Tags.....	64
Specifying a Message I/O Tag.....	65
Setting I/O Access Parameters .....	66
Retrieving Information About I/O Tags at Run Time.....	67
IOGetNode() Function .....	67
IOGetApplication() Function.....	67
IOGetTopic() Function .....	68
Dynamically Changing I/O Tag References at Run Time .....	68
.Reference Dotfield.....	68
IOSetItem() Function .....	69
IOSetAccessName() Function .....	70
<b>Converting Tags to Remote References.....</b>	<b>71</b>
<b>Accessing I/O Data by Remote References.....</b>	<b>74</b>
Redirecting Remote References During Run Time.....	75
IOSetRemoteReferences() Function .....	75
Restoring References.....	77
<b>Accessing Application Server Data from InTouch.....</b>	<b>78</b>
Using Application Server Object Attributes with InTouch Tags .....	78
Browsing Application Server Object Attributes from InTouch.....	79
Application Server Browser Restrictions.....	79
Special Extensions in Application Server Objects.....	80
Mapping Application Server Data Types to InTouch Data Types .....	81
Read/Write Behavior of Application Server Attributes.....	83
Configuring the InTouch HMI to Use a Galaxy as a Remote Tag Source.....	85
<b>Viewing Timestamp and Quality Information for an I/O Tag.....</b>	<b>88</b>
Viewing Timestamp Information for an I/O Tag .....	88
.TimeDate Dotfield.....	88
.TimeDateString Dotfield .....	89
.TimeDateTime Dotfield .....	89
.TimeDay Dotfield .....	90
.TimeHour Dotfield .....	90
.TimeMinute Dotfield .....	91
.TimeMonth Dotfield .....	91
.TimeMsec Dotfield.....	92
.TimeSecond Dotfield .....	92
.TimeTime Dotfield .....	93
.TimeTimeString Dotfield.....	93

.TimeYear Dotfield .....	94
Viewing Quality Information for an I/O Tag .....	94
Quality Data Format .....	94
About Data Quality Dotfields .....	95
.Quality Dotfield .....	95
.QualityLimit Dotfield .....	96
.QualityLimitString Dotfield .....	97
.QualityStatus Dotfield .....	97
.QualityStatusString Dotfield .....	98
.QualitySubstatus Dotfield .....	98
.QualitySubstatusString Dotfield .....	99
<b>Initializing and Resetting I/O Connections at Run Time .....</b>	<b>99</b>
Reinitializing I/O Connections with Commands .....	100
Reinitialize I/O Connections with Scripts .....	102
IOReinitAccessName() Function .....	102
IOReinitialize() Function .....	103
IOStartUninitConversations() Function .....	103
<b>Using Failover Functionality with Access Names .....</b>	<b>104</b>
Configuring Failover .....	104
Editing the Access Name Parameters of a Failover Pair .....	106
Removing Failover for an Access Name .....	106
Forcing Failover to a Backup Access Name .....	106
Failover Expression .....	107
IOForceFailover() Function .....	107
Temporarily Disabling Failover Functionality .....	107
Disable Failover Configuration Option .....	108
IODisableFailover() Script Function .....	108
Retrieving Information About Failover Pairs Using Scripting .....	109
IOGetAccessNameStatus() Function .....	109
IOGetActiveSourceName() Function .....	110
<b>Monitoring the Status of an I/O Connection .....</b>	<b>111</b>
Using IOStatus Topic Name .....	111
Using IOStatus Topic Name in Excel .....	113
Monitoring I/O Server Communications Status .....	113
<b>Accessing InTouch Tag Data from Other Applications .....</b>	<b>114</b>
<b>Chapter 6 Defining Indirect Tags .....</b>	<b>115</b>
About Defining Indirect Tags .....	115
Using Indirect Tags with Scripts .....	115
Using Indirect Tags with Local Tags .....	116
Using Indirect Tags with Remote References .....	116
<b>Chapter 7 Defining Reusable Tag Structures .....</b>	<b>119</b>
About Defining Reusable Tag Structures .....	119

<b>Defining a SuperTag Template</b> .....	<b>120</b>
Editing SuperTag Templates and Member Tags.....	122
<b>Creating Instances of SuperTags</b> .....	<b>123</b>
Using the Tagname Dictionary to Create a SuperTag Instance.....	123
Using the Tagname Dictionary to Replicate a SuperTag Instance.....	124
Using the Tagname Dictionary to Add a Tag to a SuperTag Instance .....	125
Other Ways to Create SuperTags .....	126
<b>Referencing SuperTag Members</b> .....	<b>126</b>
Importing SuperTags with the Bulk Import Utility .....	127
<b>Chapter 8 Reducing Tag Usage</b> .....	<b>128</b>
<b>Reducing Tag Usage About Reducing Tag Usage</b> .....	<b>128</b>
<b>InTouch and the Historian</b> .....	<b>128</b>
<b>Determining Tag Usage</b> .....	<b>129</b>
Determining Tag Counts.....	129
Determining Maximum Number of Remote Tags Based on Licensing.....	130
Locating Where Tags are Used.....	131
Understanding the Cross Reference Utility Report .....	131
Including Graphics from the Graphic Toolbox.....	133
Cross Reference Tag Usage for InTouch Tags in ArchedrA or Situational Awareness Library Symbols.....	133
Displaying Tag Usage in the Cross Reference Utility .....	134
Saving and Printing a Tag Cross-Reference List.....	136
<b>Deleting Unused Tags</b> .....	<b>138</b>
<b>Chapter 9 Recording Tag Values</b> .....	<b>141</b>
<b>Recording Tag Values About Recording Tag Values</b> .....	<b>141</b>
<b>Configuring Historical Logging</b> .....	<b>142</b>
Configuring Tags for Historical Logging.....	142
Configuring General Logging Properties - Historical Log File .....	143
Configuring General Logging Properties - Storage to Historian .....	145
Configuring the Affix String .....	148
Controlling Historical Logging Frequency.....	149
<b>Starting and Stopping Historical Logging at Run Time</b> .....	<b>149</b>
<b>Chapter 10 Trending Tag Data</b> .....	<b>151</b>
<b>About Trending Tag Data</b> .....	<b>151</b>
<b>Types of InTouch Trends</b> .....	<b>151</b>
Understanding Historical Trends.....	152
Understanding Real-Time Trends.....	152
<b>Showing Saved Tag Values in a Historical Trend</b> .....	<b>153</b>
<b>Using Historical Trend Objects</b> .....	<b>153</b>
Creating a Historical Trend.....	153
Configuring Which Tags to Show From a Historical Trend.....	154

Configuring the Time Span of a Historical Trend .....	155
Configuring Historical Trend Display Options .....	156
Changing the Trend Configuration at Run Time.....	157
Controlling a Historical Trend Using Dotfields .....	159
.DisplayMode Dotfield .....	159
.MinRange Dotfield.....	160
.MaxRange Dotfield .....	160
.UpdateCount Dotfield .....	161
.UpdateInProgress Dotfield .....	162
.UpdateTrend Dotfield.....	163
.ChartLength Dotfield .....	163
.ChartStart Dotfield.....	164
.Pen1-8 Dotfields .....	165
.TagID Dotfield.....	166
.ScooterLockLeft Dotfield .....	167
.ScooterLockRight Dotfield .....	168
.ScooterPosLeft Dotfield.....	168
.ScooterPosRight Dotfield.....	169
<b>Using the Historical Trend Wizard.....</b>	<b>170</b>
Creating a Trend With the Historical Trend Wizard .....	171
Configuring Which Tags to Display on the Trend Graph .....	171
Configuring the Historical Trend Time Span.....	172
Configuring Display Options.....	173
Changing the Configuration at Run Time .....	174
<b>Controlling a Historical Trend Wizard Using Scripts .....</b>	<b>174</b>
Updating the Trend to the Current Time .....	175
HTUpdateToCurrentTime() Function.....	175
Changing the Trend Configuration .....	175
HTSelectTag() Function.....	175
HTSetPenName() Function .....	176
Retrieving Information About the Trend and Historical Data .....	177
HTGetPenName() Function.....	177
HTGetTimeAtScooter() Function .....	178
HTGetTimeStringAtScooter() Function.....	178
HTGetValue() Function .....	179
HTGetValueAtScooter() Function .....	180
HTGetValueAtZone() Function.....	181
Panning and Zooming the Trend.....	182
HTScrollLeft() Function .....	182
HTScrollRight() Function .....	182
HTZoomIn() Function.....	183
HTZoomOut() Function.....	184
Printing the Trend .....	184
PrintHT() Function .....	185
Troubleshooting the Trend .....	185
HTGetLastError() Function.....	185
Displaying Real-Time Values in a Trend .....	186



<b>Using Real-Time Trend Objects .....</b>	<b>186</b>
Creating a Real-Time Trend .....	186
Configuring Which Tags to Display on a Real-time Trend .....	187
Configuring the Real-Time Trend Time Span and Update Rate .....	188
Configuring Real-time Trend Display Options .....	188
<b>Printing a Trend at Run Time .....</b>	<b>190</b>
Configuring Trend Printing Options .....	190
<b>Displaying Historical Tag Values from Other InTouch Nodes or IndustrialSQL Server .....</b>	<b>192</b>
Using the InTouch HMI with the IndustrialSQL Server .....	192
Configuring Pens to Display Remote Trend Data .....	193
Using the Tag Browser to Assign Pens to Remote History Providers .....	194
Using a QuickScript to Assign a Pen to a Remote History Provider .....	194
<b>Chapter 11 Accessing Historical Tag Values from Other Applications .....</b>	<b>196</b>
<b>About Accessing Historical Tag Values from Other InTouch Applications .....</b>	<b>196</b>
<b>Using DDE Items to Show Historical Data .....</b>	<b>197</b>
<b>Accessing Log Data with DDE .....</b>	<b>199</b>
Manually Extracting Log Data with HistData .....	199
Create a HistData Access Name .....	199
Create HistData Tags .....	200
Create a HistData Window .....	201
Run HistData .....	203
Using the HistData Wizard to Extract Log Data .....	203
<b>Accessing Historical Data from Other Applications .....</b>	<b>204</b>
<b>Troubleshooting HistData Errors .....</b>	<b>205</b>
<b>Appendix A IEEE Decimal Units .....</b>	<b>208</b>
<b>IEEE Decimal Units About IEEE Decimal Units .....</b>	<b>208</b>
<b>Showing Floating Point Numbers in the Historian HMI .....</b>	<b>208</b>
<b>Appendix B InTouch Licensing .....</b>	<b>210</b>
<b>InTouch Licensing About InTouch Licensing .....</b>	<b>210</b>
<b>Understanding License Tag Counts .....</b>	<b>210</b>
Understanding InTouch Remote Reference Limits .....	211
<b>Remote Tag Count Functions .....</b>	<b>212</b>
IORRGetSystemInfo() Function .....	213
IORRWriteState() Function .....	214
IORRGetItemActiveState() Function .....	216
<b>Index .....</b>	<b>218</b>

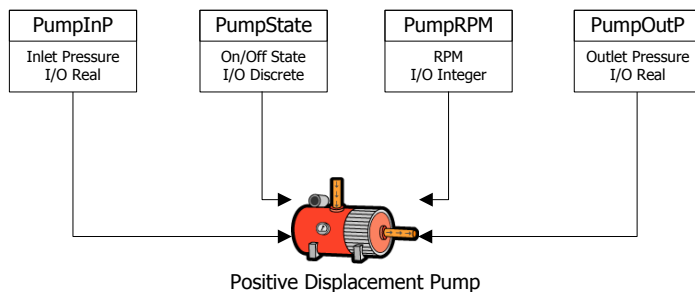
## Chapter 1

# Data Management Overview

## About Data Management

A InTouch® Human Machine Interface (HMI) application is a graphical representation of the components in a manufacturing environment. Plant operators work with this graphical interface to monitor and administer their manufacturing processes.

The figure below shows an example of a pump that is a component of a manufacturing process. The pump has properties with associated values. Pressure, RPM, and status are pump properties whose values are monitored through an HMI.

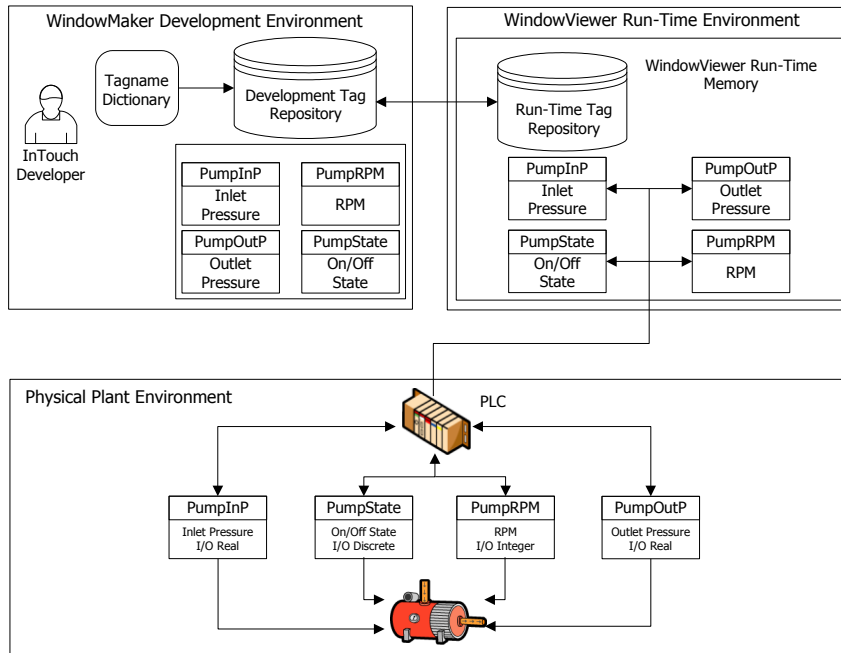


A *tag* represents a data item in an InTouch HMI application. You use tags to make specific component properties accessible as data items from a manufacturing environment. In the figure above, the PumpState tag indicates whether the pump is on or off. You create tags for components in your manufacturing environment whose properties you want to monitor or control in your InTouch application.

You can use different types of tags for the different types of data collected from a manufacturing component. For example, the PumpState tag returns a Boolean On/Off value to indicate if the pump is running or stopped. You assign the appropriate type of InTouch tag for the type of data that you want to be part of your application.

## Working with InTouch Tags

You start by creating an InTouch application. You define tags for the application using the Tagname Dictionary, which is a WindowMaker tool. The figure below shows the InTouch development and run-time environments.



You assign the name and type of tag with the Tagname Dictionary. For some types of tags, you have other options in the Tagname Dictionary to specify additional properties of tags. For example, I/O type tags include additional options to specify the connection to a remote data source.

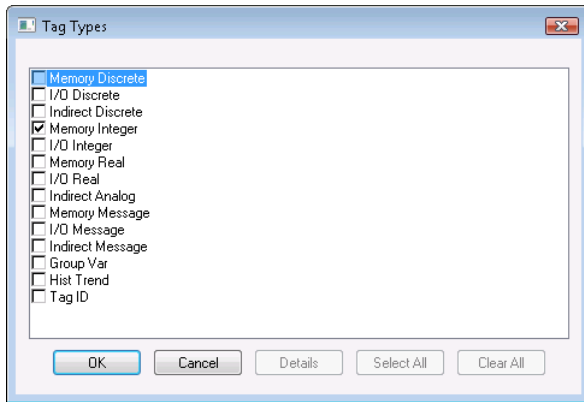
InTouch applications run in the WindowViewer environment. When WindowViewer starts an application, it reads the tags from the development repository and places them into run-time memory.

The InTouch application communicates with the tags placed into run-time memory using animation links or scripts. The InTouch application tracks the current values and other status information from the component properties assigned to tags.

## Types of InTouch Tags

When you define a tag, you assign to it a specific type according to the process data that will be associated with the tag. For example, if a tag shows the RPM of a pump, then you assign the tag as an integer tag type.

In the Tagname Dictionary, you use the **Tag Types** dialog box to assign the tag type to any tag you created.



After you assign a tag type, the Tagname Dictionary lists specific options for the type of tag you selected.

## Memory Tags

Memory tags define internal system constants and variables within InTouch applications. For example, you can define an internal constant as a real number of 3.414. In process simulations, memory tags can control the actions of background QuickScripts by acting as a counter. Based upon the count associated with the tag, the QuickScript can trigger various animation effects. Memory tags can also act as calculated variables that are accessed by other programs.

Select from the four types of memory tags, based upon the process data associated with the tag.

- **Memory Discrete**

Memory discrete tags are associated with the state properties of a process component. The values assigned to memory discrete tags are two possible Boolean states such as:

- 0 or 1
- False or true
- On or off
- High or low

- **Memory Integer (Analog)**

You can assign memory integer tags 32-bit signed-integers between -2,147,483,648 and 2,147,483,647.

- **Memory Real (Analog)**

You can assign memory real tags floating decimal point numbers between  $-3.4 \times 10^{38}$  and  $3.4 \times 10^{38}$ . All floating point calculations are performed with 64-bit resolution, but the results are stored as 32-bit decimal numbers. For more information about the maximum precision of real numbers, see *IEEE Decimal Units* on page 208.

- **Memory Message**

You can assign memory message tags text strings up to 131 single-byte characters in length.

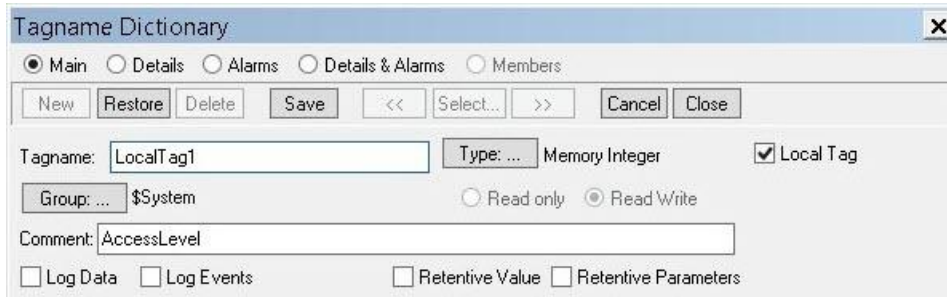
## Local Tags

Local Tags allow the user to create per-session memory tags for use in the Web Client. For example, the user can configure a local memory tag to be used for navigation. In runtime when the user clicks the navigation symbol linked to that tag, each session will behave independently.

---

**Note:** .Dot fields are not supported for local tags.

---



To create a local tag:

- Create a memory tag and select the **Local Tag** checkbox. Configure other tag properties. For more information on Creating New Tags, see *Creating New Tags* on page 22
- Click **Save**.

## I/O Tags

I/O tags read or write InTouch application data to or from an external source. External data includes input and output from programmable controllers, process computers, and network nodes. I/O tag data values are remotely accessed through the following protocols:

- Microsoft Dynamic Data Exchange (DDE)
- SuiteLink™

When the value of an I/O tag changes in run-time memory, the InTouch HMI updates the remote application. Conversely, I/O tag values in InTouch are updated whenever the values of corresponding data items change in a remote application.

The InTouch HMI provides four types of I/O tags based upon the process data associated with the tag. These four types of I/O tags are similar to memory tag types.

- **I/O Discrete**  
I/O discrete tags are associated with component process properties whose values are represented by two possible states such as:
  - 0 or 1
  - False or true
  - On or off
  - High or low
- **I/O Integer (Analog)**

I/O integer tags can be assigned 32-bit signed-integers between -2,147,483,648 and 2,147,483,647.

- **I/O Real (Analog)**

I/O real tags can be assigned floating decimal point numbers between  $-3.4 \times 10^{38}$  and  $3.4 \times 10^{38}$ . All I/O real tag floating point calculations are performed with 64-bit resolution, but the results are stored as 32-bit numbers. For more information about the maximum precision of I/O real numbers, see *IEEE Decimal Units* on page 208.

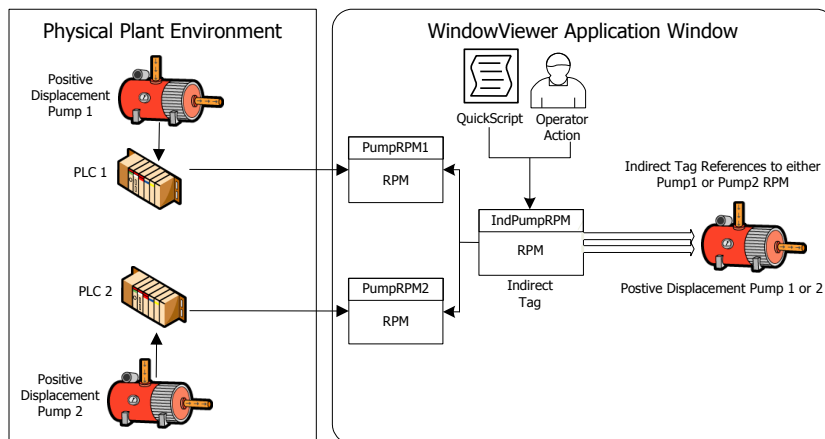
- **I/O Message**

I/O message tags can be assigned text strings up to a maximum of 131 single-byte characters.

## Indirect Tags

Indirect tags act as "pointers" to other tags. For example, you can create a single InTouch window and use indirect tags to show data from multiple different sets of tags.

The following figure shows an example of an application window that is capable of displaying several pumps. Instead of creating separate windows for each pump, you can use indirect tags in one window to show the values of different source tags associated with individual pumps.



A QuickScript or operator action points the indirect tag to the source tags. For example, the following script statements assign the two PumpRPM tags to an indirect analog tag called **IndPumpRPM** based on the value of the PumpNo tag.

```
IF PumpNo == 1 THEN
    IndPumpRPM.Name = "PumpRPM1";
ELSE
    IndPumpRPM.Name = "PumpRPM2";
ENDIF;
```

When you equate an indirect tag to another source tag, the indirect tag acts as if it is the source tag. The values associated with the original source and indirect duplicate tags are synchronized together. If the value of the source tag changes, the indirect tag reflects the change. If the indirect tag's value changes, the source tag changes accordingly.

You can use discrete, analog, and message types of indirect tags. These three types of indirect tags are comparable to similar memory and I/O types of tags.

For more information about indirect tags, see *Defining Indirect Tags* on page 115.

## Miscellaneous Tags

You can use other types of InTouch tags designed for specific, restricted purposes. You can use these tags to create dynamic alarm displays, create historical trends, and change the tags assigned to historical trend pens.

### Hist Trend tags

Hist Trend tags can be used to reference historical trend graphs. All of the **dotfields** associated with historical trends can be applied to **Hist Trend** tags.

For more information about defining and using Hist Trend tags, see *Trending Tag Data* on page 151.

### Tag ID Tags

Tag ID tags retrieve information from tags whose values are plotted in an InTouch historical trend graph. Typically, you use a Tag ID tag to show the name of the tag assigned to a specific trend pen or change the tag assigned to a trend pen.

You can include a statement in a QuickScript to assign a new tag to a pen in any historical trend using a Tag ID type tag. For example, the following QuickScript statement changes the tag associated with a historical trend pen:

```
HistTrend.Pen1=MyLoggedTag.TagID;
```

When this QuickScript runs, Pen1 of the historical trend begins trending the historically logged data for the MyLoggedTag.

For more information about defining and using Hist Trend tags, see *Using the Historical Trend Wizard* on page 170.

### SuperTags

A SuperTag is a template that contains a set of related tags. For example, you can create a SuperTag template containing a set of tags assigned to all the properties of a pump.

Use SuperTags when you have identical equipment in your production process. Instead of creating a set of tags for each piece of equipment, assign an instance of the SuperTag template to each of the identical process items.

For more information about SuperTags, see *Defining Reusable Tag Structures* on page 119.

### Obsolete Tags

Using a Group Var tag, you can create dynamic alarm displays, dynamic logging on disk, and dynamic printing with the standard alarm system of InTouch. Group Var tags are included only for backward compatibility with applications developed with InTouch version 7.11 and earlier. Do not use Group Var tags in applications developed with InTouch versions later than 7.11.

## Tag Properties

Each InTouch tag type has a set of properties that describe the characteristics of data associated with the tag. The four principal data types associated with InTouch tags are:

- discrete values
- integer numbers
- real numbers
- text messages

All tag properties are assigned initial values when you create a tag with the Tagname Dictionary. The following figure shows the properties of an I/O Integer tag.

The screenshot shows the 'Tagname Dictionary' dialog box with the following details:

- Tagname:** PumpRPM
- Type:** I/O Integer
- Group:** \$System
- Access:** Read/Write
- Comment:** AccessLevel
- Initial Value:** 0
- Min EU:** -32768
- Max EU:** 32767
- Deadband:** 0
- Min Raw:** -32768
- Max Raw:** 32767
- Eng Units:** (empty)
- Log Deadband:** 0
- Conversion:** Linear
- Item:** RPM
- ACK Model:** Condition
- Alarm Comment:** (empty)
- Alarm Settings:** LoLo, Low, Minor Deviation, Major Deviation, Rate of Change, High, HiHi, and Deviation Deadband % are all set to 0.
- Priority:** 1
- Alarm Inhibitor:** (empty)

After setting the initial values of tag properties from the Tagname Dictionary, you can dynamically change most tag properties using dotfields. A dotfield identifies a tag property that can be monitored or modified by a script when the InTouch application is running. You append the dotfield to the name of the tag in a script.

For more information about using dotfields to dynamically change tag properties, see *Using Tag Dotfields to View or Change Tag Properties* on page 38.

## Memory Tag Properties

The following table lists the properties of the four types of memory tags. Each property can be selected or modified as an option of the Tagname Dictionary. For more information, see *Creating New Tags* on page 22.

Tag Properties	Discrete	Integer	Message	Real
% Deviation		•		•
% per		•		•



Tag Properties	Discrete	Integer	Message	Real
ACK Model	•	•		•
Alarm Comment	•	•	•	•
Alarm Group	•	•	•	•
Alarm Inhibitor	•	•		•
Alarm State	•			
Alarm Value		•		•
Comment	•	•	•	•
Deadband		•		•
Eng Units		•		•
Deviation Deadband %		•		•
High		•		•
HiHi		•		•
Initial Value	•	•	•	•
Log Data	•	•		•
Log Deadband		•		•
Log Events	•	•	•	•
Lo		•		•
LoLo		•		•
Maximum Length			•	
Major Deviation		•		•
Max Value		•		•
Min Value		•		•
Minor Deviation		•		•
Off Msg	•			
On Msg	•			
Priority	•	•	•	•
Rate of Change		•		•
Read Only	•	•	•	•

Tag Properties	Discrete	Integer	Message	Real
Read Write	•	•	•	•
Retentive Parameters		•		•
Retentive Value	•	•	•	•
Target		•		•
Value Deadband		•		•

## I/O Tag Properties

Like memory tags, I/O tag properties can be selected or modified as options in the Tagname Dictionary. For more information, see *Creating New Tags* on page 22.

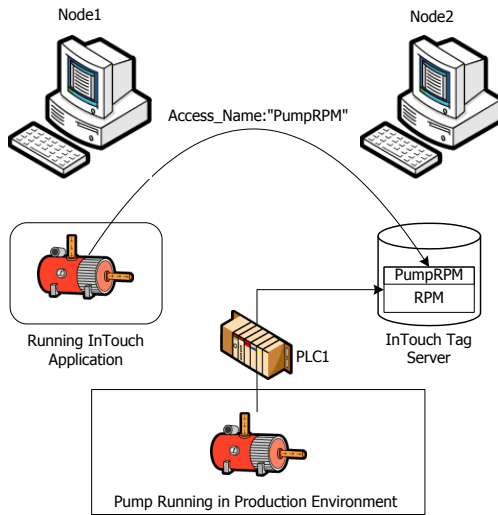
Tag Properties	Discrete	Integer	Message	Real
% Deviation		•		•
% per		•		•
Access Name	•	•	•	•
ACK Model	•	•		•
Alarm Comment	•	•	•	•
Alarm Group	•	•	•	•
Alarm Inhibitor	•	•		•
Alarm State	•			
Alarm Value		•		•
Comment	•	•	•	•
Conversion		•		•
Deadband		•		•
Deviation Deadband %		•		•
Eng Units		•		•
High		•		•
HiHi		•		•
Initial Value	•	•	•	•

Tag Properties	Discrete	Integer	Message	Real
Input Conversion	•			
Item	•	•	•	•
Log Data	•	•		•
Log Deadband		•		•
Log Events	•	•	•	•
Lo		•		•
LoLo		•		•
Maximum Length			•	
Major Deviation		•		•
Max EU		•		•
Max Raw		•		•
Max Value		•		•
Min EU		•		•
Min Raw		•		•
Min Value		•		•
Minor Deviation		•		•
Off Msg	•			
On Msg	•			
Priority	•	•	•	•
Rate of Change		•		•
Read Only	•	•	•	•
Read Write	•	•	•	•
Retentive Parameters		•		•
Retentive Value	•	•	•	•
Square Root Conversion		•		•
Target		•		•
Use Tagname as Item Name	•	•	•	•

Tag Properties	Discrete	Integer	Message	Real
Value Deadband		•		•

## Remote Tag References

You can create distributed InTouch applications with a tag server running on a separate node from the node running the InTouch application. The following figure shows an InTouch application that makes a remote reference to the PumpRPM tag from a tag server running on another node.



You create an InTouch application to reference tags located on a remote node by two methods:

- Associate I/O tags with an Access Name that identifies a remote server as the tag source. For more information about defining an Access Name for an I/O tag, see *Setting Up Access Names* on page 61.
- Use a remote reference directly to **the tag**. For example, PLC1:PumpRPM.

For more information, see *Accessing I/O Data by Remote References* on page 74.

## Chapter 2

# Managing Tags with the Tagname Dictionary

## About Managing Tags with the Tagname Dictionary

Using the Tagname dictionary, you create tags for an InTouch application. The figure below shows the **Tagname Dictionary** dialog box with all options to define the properties of an I/O tag.



## Planning Tag Usage

You can reduce development time by identifying the key requirements of an application's tags in a preliminary planning phase. Thorough planning reduces the time you need to create InTouch applications.

Before creating tags:

- Identify all physical components of the process that need to be represented in the InTouch application. Create a list of component attributes that need to be represented as data sources in the application.
- Identify the type of data associated with each component attribute.

Assign each tag a tag type based upon the data associated with the component attribute. For more information about assigning a data type to a tag, see *Creating New Tags* on page 22.

- Determine the characteristics of data that needs to be incorporated into your InTouch application. Assess the following data characteristics for each tag:

- Expected range of data values
- Units of measure assigned to data values
- Initial data value
- Deadband value to set a threshold when a tag's value is recognized as changed
- Messages to be shown when a tag's value changes state

For more information about defining the characteristics of tag data, see *Understanding Tag Properties* on page 25.

- Develop a tag naming convention and standard.

Typically, complex applications require many tags. Develop a standardized naming convention that suggests the organization of the tags within the application. For more information about tag naming conventions, see *Tag Name Conventions* on page 24.

- Determine what process data needs to be saved.

Selected data is saved to a log file. You can use logged data to create historical trends that show the changes in tag values over time. For more information about setting tag logging, see *Tag Logging* on page 27.

## Creating New Tags

You create tags with the WindowMaker **Tagname Dictionary**. Before you start, analyze your plant process to determine the tags that you need to create in your InTouch application.

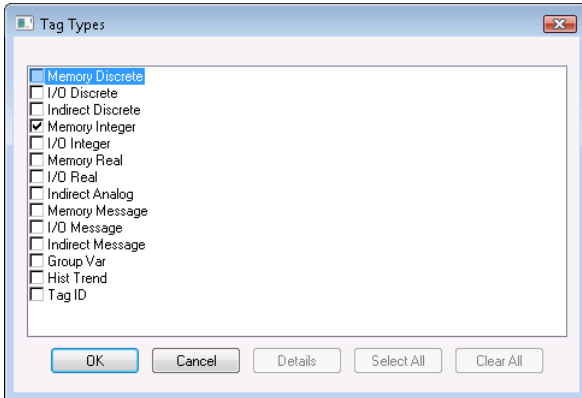
### To create a new tag

1. Open the InTouch application in WindowMaker.
2. On the **Special** menu, click **Tagname Dictionary**.

The first time you open the Tagname Dictionary, the definition for the **\$AccessLevel system tag** appears in the **Tagname** box. After saving a new tag, the **Tagname Dictionary** shows the most recently saved tag definition.

3. Do the following:
  - a. Click **New**. The **Tagname** box clears.
  - b. Enter a name for the new tag. For more information about tag naming requirements, see *Tag Name Conventions* on page 24.

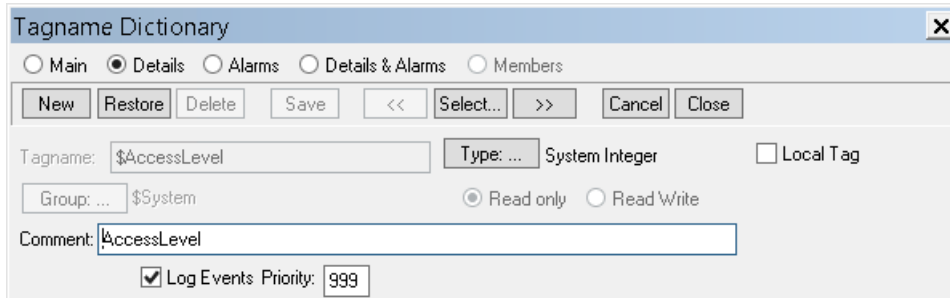
- c. Optionally, type a comment about the new tag in the **Comment** box.
- 4. Click **Type**. The **Tag Types** dialog box appears with a list of supported InTouch tag types.



- 5. Select a type of tag from the list and click **OK**. The Tagname Dictionary reappears and shows the type of tag you selected.
- 6. If needed, click **Details** to see the additional Tagname Dictionary options for the selected tag type.
- 7. Specify further tag options in the **Tagname Dictionary** dialog box.  
For more information about specifying tag properties, see *Configuring Tag Properties* on page 23.
- 8. Click **Save**. Click **Close** to close the **Tagname Dictionary** dialog box.

## Configuring Tag Properties

Using the **Tagname Dictionary** dialog box, you can specify common tag properties that are part of every tag definition. You must assign a name to each tag. You can add an optional comment. Both the tag name and comment are common properties of all tags.



Each type of InTouch tag has unique data properties. After you select a tag type, the **Tagname Dictionary** dialog box expands to show a set of options, based upon the selected tag type.

## Common Tag Properties

You must assign a unique name to each tag. An optional comment can be part of the tag definition. It is good practice to assign an appropriate comment for each tag you define.

All tags belong to an alarm group, which is another common tag property. By default, all tags belong to the \$System alarm group. For more information about assigning tags to other alarm groups, see *Configuring Alarms* in the InTouch® HMI Alarms and Events Guide.

## Tag Name Conventions

When you name your tags, use a consistent naming convention if you need many tags with similar properties.

Follow these naming conventions for InTouch tag names:

- Use 32 characters or fewer in a tag name.
- Use an alphanumeric (**A-Z, a-z, 0-9**) as the first character of a tag name.

A best practice is to use only alphanumeric characters for tag names.

- Use at least one alphabetic character in the tag name.

(Optional) Use the following special characters:

- Dash	! Exclamation point	# Number sign
\$ Dollar sign	% Percent	& Ampersand
? Question mark	@ At sign	_ Underscore

If possible, avoid using special characters in tag names unless absolutely required by your application.

- Avoid using a dash (-) in a tag name.

A dash is a valid character for an InTouch tag name. But, InTouch evaluates a dash as a negation or subtraction operator in logical or arithmetic expressions. For example, the expression  $A=B-C$  can be interpreted as  $A=B$  minus  $C$  or the tag named  $B-C$  is assigned to tag  $A$ .

Hyphen or minus sign (-) will not be allowed when Tag Names start with a numeric character.

- Do not use blank spaces in tag names.
- Do not use a number in the tag name that can be interpreted as an exponential number.

For example, you cannot name a tag  $125E4$  because it could be interpreted as a base number with an exponent raised to the fourth power.

- Do not use a number in the tag name that can be interpreted as a hexadecimal number.

For example, you cannot name a tag  $0x123B$  because it could be interpreted as a hexadecimal number.

## Automatically Naming Tags

As you name your tags in the Tagname Dictionary, the InTouch HMI tracks the naming conventions you are using. If you name your tags Pump01, Pump02, the InTouch HMI suggests the next tag name as Pump03. You can accept or reject this name. This naming help is called Indexing.

Indexing is based on the last consecutive number in a tag name. For example, if your tag name is PumpInP04LotB99A, the InTouch HMI suggests the next tag name as PumpInP04LotB100A, not PumpInP05LotB99A.



## Tag Comments

You can enter an optional comment up to 50 characters in the Tagname Dictionary **Comment** box when you create a tag.

The first time you access the Tagname Dictionary, the default comment for the **\$AccessLevel** system tag appears in the **Comment** box. Delete this comment to prevent it from being associated with any tags that you create.

## Understanding Tag Properties

After specifying common tag properties, you must define other properties that are specific to the type of tag you are creating. The following table shows the basic properties for memory tags by tag type.

Tag Type	Unique Properties
Discrete	Initial Value, On Msg, Off Msg, Comment
Integer	Initial Value, Min Value, Deadband, Eng Units, Max Value, Log Deadband, Comment
Real	Initial Value, Min Value, Deadband, Eng Units, Max Value, Log Deadband, Comment
Message	Maximum Length, Initial Value, Comment

I/O tags have additional properties to establish network communication and transform raw data from network devices to normalized values used by the InTouch application. For more information about defining I/O tags, see *Configuring I/O Tag Properties* on page 63.

## Value Ranges, Measurement Units, and an Initial Value

Discrete, integer, real, and message tags are assigned an initial value when the InTouch application starts in WindowViewer. In the case of a discrete tag, the initial value is one of the possible binary states. For integer and real tags, the initial value is the number associated with the tag when the application starts. The initial value is specified in the Tagname Dictionary.

You can specify the initial tag value to be the last value of the tag when the application stops running in WindowViewer. By selecting the **Retentive Value** option from the Tagname Dictionary, the tag is assigned its last active value as the initial value when the application starts again.

Integer and real tags include properties that set the lower and upper boundaries of the range of possible numerical values assigned to a tag. Both integer and real tags include the **Min Value** and the **Max Value** properties that define the lower and upper limits of the range.

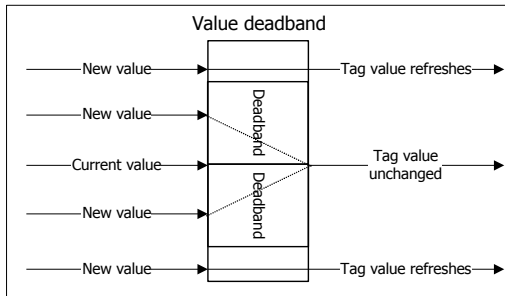
Integer and real tags also include the **Eng Units** property to assign an engineering units label that describes the unit of measure for the tag's value. For example, you can assign PSI as the **Eng Units** property for an integer tag associated with pump pressure.

## Tag Deadbands

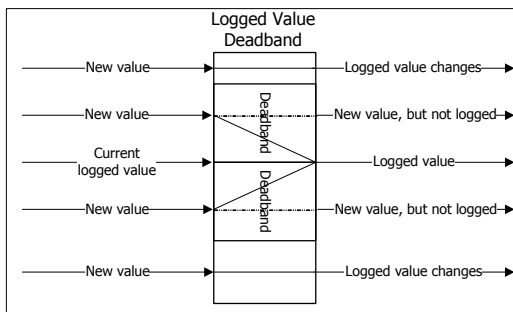
A deadband is a sensitivity setting for tag values. A deadband is usually associated with I/O tags whose values change constantly. A deadband filters out small momentary changes in a tag value to reduce the amount of InTouch data processing.

The Tagname Dictionary includes two deadband properties for tags associated with integer and real data.

- **Value Deadband:** The value deadband property sets a threshold that must be exceeded before WindowViewer refreshes the tag's value in run-time memory. The following figure shows the absolute deadband range around a current tag value.



- **Log Value Deadband:** The log deadband sets a threshold that must be exceeded before the tag's value is written to the log file. The following figure shows the deadband around the current value of a logged tag.



Only new tag values outside of the deadband are written to the log file. Small value changes within the deadband range are ignored and not logged.

### To set a tag deadband

1. Open the **Tagname Dictionary** dialog box.
2. Click **Select**. The **Select Tag** dialog box appears. The tags currently defined for the application are listed.
3. Select an integer or real tag type from the list.
4. Click **OK**. The detail portion of the **Tagname Dictionary** dialog box shows additional options when you select a real or integer tag.

Initial Value:	<input type="text" value="0"/>	Min Value:	<input type="text" value="0"/>	Deadband:	<input type="text" value="15"/>
Eng Units:	<input type="text" value="RPM"/>	Max Value:	<input type="text" value="2500"/>	Log Deadband:	<input type="text" value="25"/>

5. Set the value deadband by entering an integer or real number in the **Deadband** box based upon the selected tag type.

The value deadband sets an absolute threshold level in engineering units.

6. Set the log deadband by entering an integer or real number in the **Log Deadband** box based upon the selected tag type.

Like the value deadband, the log deadband sets an absolute threshold in engineering units.

7. Click **Save** to save your deadband changes.
8. Click **Close** to close the Tagname Dictionary dialog box.

## Tag Value Retention

The detail portion of the **Tagname Dictionary** includes two properties to retain tag values and operator changes to alarm limits.

All tag types include a **Retentive Value** property. Select **Retentive Value** to retain the current value of the tag when the application stops. When you start the application again, WindowViewer uses the retained value as the initial value of the tag.

WindowViewer does not write retained values to I/O devices when WindowViewer starts the application again. I/O values are updated after the I/O Server initially scans the device providing data.

The **Retentive Value** option cannot be selected or cleared for new or existing tags if WindowViewer is running. When you select this option, the initial value of the tag is constantly updated to reflect its current value. When WindowViewer stops, the initial value is set to the last retained value. If this option is later cleared, the initial value of the tag is set to the last retained value.

Integer and real tags include the **Retentive Parameters** property. Select **Retentive Parameters** to retain any changes made by the operator to a tag's alarm limit while the application is running. WindowViewer uses the modified alarm limit as the initial value for the alarm limit when the application is restarted.

## I/O Connection

All types of I/O tags must identify the Access Name and Item Name of the external data source. For more information about specifying the Access Name and Item Name for I/O tags, see *Setting I/O Access Parameters* on page 66.

## Tag Logging

During run time, WindowViewer can write an entry to the historical log file each time a tag's value changes more than the specified log deadband. WindowViewer also writes entries to the log at a fixed interval regardless of current tag values. By default, this fixed interval is one hour.

---

**Note:** For more controllable and versatile logging, consider using Historian to store InTouch historical data.

---

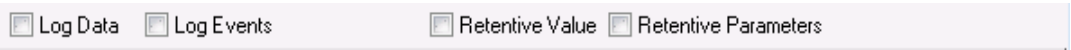
The **Tagname Dictionary** dialog box includes separate options to log data and events to the log file. You can set value logging options. For information about setting event logging, see *Configuring Alarms in the InTouch® HMI Alarms and Events Guide*.

For a tag's value to be written to the historical log file, historical logging must be enabled. For more information about setting general logging properties, see *Configuring Historical Logging* on page 142.

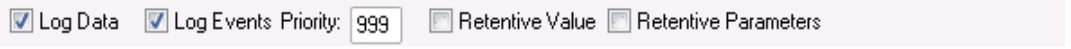
For integer and real tags, you can set the Log Deadband in their respective details dialog boxes. The **Log Deadband** option specifies how many engineering units a tag's value must change to write a log entry.

**To configure logging for a tag**

1. Open the **Tagname Dictionary**.
2. Select the tag whose data you want saved to the log file.
3. Select **Log Data**.



4. Select **Log Events** if you want to log value changes to the tag initiated by an operator, an I/O, QuickScript, or operating system. The **Priority** box appears after you select **Log Events**.



The **Priority** value determines the event priority for the tag. Valid values are 1 to 999, where 1 is the highest priority and 999 is the lowest.

5. Click **Save** and then close the Tagname Dictionary.

## Creating Discrete Tags

You can specify discrete tags to show the binary state of internal processes running in an InTouch application. A discrete tag must be assigned an initial value of on or off. Also, you can specify messages that appear in the alarm event window when the process associated with the tag transitions into or out of an alarm state.

The following steps show how to define a memory discrete tag. I/O discrete tags indicate the binary state of all inputs and outputs from programmable controllers, process computers, and data from network nodes.

For more information about setting the properties of an I/O discrete tag, see *Specifying a Discrete I/O Tag* on page 63.

**To define an initial value and messages for a memory discrete tag**

1. Select **Memory Discrete** as the type of tag in the **Tag Types** dialog box.
2. If needed, select **Details** at the top of the **Tagname Dictionary** dialog box to show the detail options. The detail portion of the **Tagname Dictionary** dialog box appears.



3. Select **On** or **Off** as the initial value associated with the tag. The tag is set to this initial value when the application starts.
4. Enter messages in the **On Msg** and **Off Msg** boxes that appear when the tag transitions in and out of an alarm state.

These messages are available for use in any animation link or script, regardless of whether the tag has alarms configured or not.

- If you define a discrete alarm that is active when the tag value is equal to 1 (On, True), the message entered in the **On Msg** box appears in the **Value** and **Limit** columns of your ActiveX alarm displays.

When the tag’s alarm state returns to normal, the message entered in the **Off Msg** box appears in the **Value** column and the On message remains in the **Limit** column.

- If you define a discrete alarm that is active when the tag value is equal to 0 (Off, False), the message entered in the **Off Msg** box appears in the **Value** and **Limit** columns of your ActiveX alarm displays.

When the tag’s alarm state returns to normal, the message entered in the **On Msg** box appears in the **Value** column and the Off message remains in the **Limit** column.

5. Save your changes to the tag.

## Creating Integer and Real Tags

You can specify integer and real tags to show numerical values of processes running in an InTouch application.

The following steps show how to define memory and I/O integer and real tags. Memory integer and real tags must be assigned an initial value. You also must set minimum and maximum data ranges.

For information about setting the I/O properties of an I/O integer and real tags, see *Specifying Integer and Real I/O Tags* on page 64.

---

**Important:** InTouch real numbers are limited to eight-digit precision. To avoid possible rounding errors, do not exceed eight-digit precision when you specify the properties of your real tags. For more information, see *IEEE Decimal Units* on page 208.

---

### To define memory integer and real tag values

1. Assign memory integer or memory real as the type of tag in the **Tag Types** dialog box. The detail portion of the **Tagname Dictionary** dialog box appears.

Initial Value:	<input type="text" value="0"/>	Min Value:	<input type="text" value="0"/>	Deadband:	<input type="text" value="15"/>
Eng Units:	<input type="text" value="PSI"/>	Max Value:	<input type="text" value="1500"/>	Log Deadband:	<input type="text" value="25"/>

2. Set the properties for integer and real tags. Do the following:
  - In the **Initial Value** box, type the integer or real number associated with the tag when the application starts.
  - In the **Min Value** box, type the minimum integer or real number for the tag.  
The **Min Value** sets the minimum possible value for numbers associated with memory integer and real tags.
  - In the **Max Value** box, type the maximum integer or real number for the tag.  
The **Max Value** sets the maximum possible value for numbers associated with memory integer and real tags.
  - In the **Eng Units** box, type the label you want to use for the tag's engineering units.
3. Save your changes to the tag.

For more information about setting the tag’s deadband and log deadband properties, see *Tag Deadbands* on page 26.

## Creating Message Tags

You can specify message tags for both internal and external processes. These tags include properties to specify an initial message that appears when WindowViewer starts the application and a comment that can be read from the Alarm Viewer.

For more information about defining an I/O message tag, see *Specifying a Message I/O Tag* on page 65.

### To define a memory message tag

1. Assign memory message as the type of tag in the **Tag Types** dialog box.
2. If needed, select **Details** to show the detail portion of the **Tagname Dictionary** dialog box.

The screenshot shows a dialog box with three input fields. The first field is labeled 'Maximum Length' and contains the value '131'. The second field is labeled 'Initial Value' and is currently empty. The third field is labeled 'Alarm Comment' and is also empty.

3. Set the properties of a memory message tag. Do the following:
  - In the **Maximum Length** box, type an integer that is the maximum number of characters that can appear in the tag's message. Or, accept the default length of 131 characters, which is the maximum message length.
  - In the **Initial Value** box, type the text of the message that you want assigned to the tag when WindowViewer starts the application.
  - In the **Alarm Comment** box, type a message that can be read in the AlarmViewer control if the message tag has the **Log Events** option selected.
4. Save your changes to the tag.

## Creating I/O Tags

I/O tags have a set of common properties that specify network connectivity between the InTouch application and external processes. The detail portion of the **Tagname Dictionary** dialog box includes options to set an I/O tag's external properties.

For more information about setting I/O properties, see *Data Access with I/O* on page 56.

## Modifying Tags

Modifying a tag is similar to creating a tag. You select the tag to be modified from the Tagname Dictionary. Then, you change the tag's properties following the same steps to create a tag.

---

**Important:** It is not easy to modify a tag's type after it is used in an InTouch application. The tag type selection may be limited to similar data types. You should carefully select the correct data type when you define your tags.

---

### To modify a tag

1. Open the **Tagname Dictionary** dialog box.
2. Click **Select**. The **Select Tag** dialog box appears. A list of tags currently defined for the application is shown.
3. Select the tag to be modified from the list.

4. Click **OK**. The **Tagname Dictionary** dialog box shows the values specified for the selected tag.
5. Make changes to the tag's properties.
6. Click **Save** to update the tag with your changes.
7. Click **Close** to close the Tagname Dictionary.

## Deleting Tags

A count is maintained for all tags defined for an InTouch application. The tag count does not decrease automatically when a window containing animation links or scripts is deleted. The tags associated with the deleted window are still considered to be in use and cannot be deleted.

To be able to delete a tag that is no longer used, you must close WindowViewer and update local and remote tag use counts. You can determine where a tag is being used with the InTouch Cross-Reference utility. For more information about using the Cross Reference Utility and updating tag counts, see *Reducing Tag Usage* on page 128.

Tags can be deleted after the use count is updated. For more information, see *Deleting Unused Tags* on page 138.

## Printing a Tag List and Usage Information

You can print the contents of an InTouch application's database, windows, and scripts in WindowMaker using the WindowMaker Printout utility.

For more information, see *Saving and Printing a Tag Cross-Reference List* on page 136.

## Chapter 3

# System Tags

## About System Tags

Using system tags provides system-related information and standard functions like date and time for InTouch scripting. System tags are part of all applications. You can also use system tags in scripts to manage a running application like:

- Monitoring and managing application security.
- Establishing I/O communications to a remote node.
- Detecting when an alarm occurs.
- Starting and stopping historical logging.
- Updating an application to a new version with NAD.

System tags are identified by a dollar sign (\$) as the first character of a tag's name in the Tagname Dictionary. System tags cannot be deleted. You can only change the comment associated with the system tag.

## System Tags Reference

The InTouch system tags are described in the following table:

System Tag	Purpose	For More Information
<b>\$AccessLevel</b>	Read-only integer tag that specifies the access level associated with the currently logged-on operator.  This information can be used in animation links or scripts to control the operator's access to specific InTouch functions.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.



System Tag	Purpose	For More Information
<b>\$ApplicationChanged</b>	Read-only discrete tag that indicates whether the master application has changed in a NAD environment.	See <i>Distributing Applications</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$ApplicationVersion</b>	Read-only real tag that specifies the current version number of the application running in WindowViewer.	See <i>Distributing Applications</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$ChangePassword</b>	Discrete write-only tag that shows the <b>Change Password</b> dialog box when set to 1.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$ConfigureUsers</b>	Write-only discrete tag that shows the generic <b>Configure Users</b> dialog box to edit the security user name list.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$Date</b>	Read-only integer tag that shows the whole number of days that have passed since January 1, 1970.	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$DateString</b>	Read-only message tag that shows the date in the same format specified from the Windows <b>Regional and Language Options</b> dialog box.	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$DateTime</b>	Read-only real tag that shows the fractional number of days that have passed since January 1, 1970.	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$Day</b>	Read-only integer tag that shows the current day of the month (1-31).	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.

System Tag	Purpose	For More Information
<b>\$False</b>	<p>Discrete read-only tag that returns a FALSE value within an expression.</p> <p>The \$False system tag is used to replace any instance of obsolete system tags when updating applications from earlier versions of InTouch to the current version.</p>	No further information.
<b>\$HistoricalLogging</b>	Read/write discrete tag used to start and stop historical logging while an InTouch application is running.	See <i>Recording Tag Values</i> on page 141.
<b>\$Hour</b>	Read-only integer tag that shows the current hour as a value from 0 to 23.	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$InactivityTimeout</b>	Read-only discrete tag that indicates the user inactivity period has elapsed. When set to 1, the inactivity period has elapsed and the user is automatically logged off from WindowViewer.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$InactivityWarning</b>	Read-only discrete tag that indicates the inactivity warning period has elapsed. The value of \$InactivityWarning can be used to issue an inactivity warning to an operator.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$Language</b>	Read/write integer tag that specifies the language ID of the language shown in an InTouch application.	See <i>Switching a Language at Run Time</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$LogicRunning</b>	Read/write discrete tag that can be used to start and stop an application script.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$Minute</b>	Read-only integer tag that shows the current minute (0-59).	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.

System Tag	Purpose	For More Information
<b>\$Month</b>	Read-only integer tag that shows the number of the current month (1-12).	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$Msec</b>	Read-only integer tag that shows the current millisecond (0-999).	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.
<b>\$NewAlarm</b>	Read/write discrete tag that indicates when a new local alarm has occurred.	See <i>Controlling Alarm Properties of Tags and Groups at Run Time</i> in the InTouch® HMI Alarms and Events Guide.
<b>\$ObjHor</b>	Read-only integer tag that shows the horizontal pixel location of the center of a selected object on the screen.	See <i>Animating Objects</i> in the InTouch® HMI Visualization Guide.
<b>\$ObjVer</b>	Read-only integer tag that shows the vertical pixel location of the center of a selected object on the screen.	See <i>Animating Objects</i> in the InTouch® HMI Visualization Guide.
<b>\$Operator</b>	Read-only message tag that shows the name of the operator logged on to an InTouch application.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$OperatorDomain</b>	Read-only message tag that contains the domain or machine name specified at log on when the application is secured with operating system-based security.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$OperatorDomainEntered</b>	Write-only message tag assigned the domain of the operator for a logon attempt to an InTouch application.  The logon attempt does not start until you assign a value to the \$PasswordEntered system tag.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.

System Tag	Purpose	For More Information
<b>\$OperatorEntered</b>	<p>Read/write message tag assigned the user account name of an operator for a logon attempt to an InTouch application.</p> <p>The logon attempt does not start until you assign a value to the \$PasswordEntered system tag.</p>	<p>See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.</p>
<b>\$OperatorName</b>	<p>Read-only message tag that shows the full name of the operator if operating system-based or ArchestrA® authentication is used.</p>	<p>See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.</p>
<b>\$PasswordEntered</b>	<p>Write-only message tag assigned the password of an operator for a logon attempt to an InTouch application.</p> <p>When you write a value to this tag, a logon attempt is started using the values of the \$OperatorDomainEntered, \$OperatorEntered, and \$PasswordEntered system tags.</p>	<p>See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.</p>
<b>\$Second</b>	<p>Read-only integer tag that shows the current second (0-59).</p>	<p>See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.</p>
<b>\$StartDdeConversations</b>	<p>Read/write discrete tag used to start uninitiated conversations during run time.</p>	<p>See <i>Data Access with I/O</i> on page 56.</p>
<b>\$System</b>	<p>Read-only tag that identifies the root alarm group.</p>	<p>See <i>Overview of Alarms and Events</i> in the InTouch® HMI Alarms and Events Guide.</p>
<b>\$Time</b>	<p>Read-only integer tag that shows the elapsed time in milliseconds since midnight of the current day.</p>	<p>See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.</p>
<b>\$TimeString</b>	<p>Read-only message tag that shows the current time in the same format specified from the Windows <b>Regional and Language Options</b> dialog box.</p>	<p>See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.</p>

System Tag	Purpose	For More Information
<b>\$VerifiedUserName</b>	Read-only message tag that contains either a verified user’s full name or null.	See <i>Securing InTouch</i> in the InTouch® HMI Application Management and Extension Guide.
<b>\$Year</b>	Read-only integer tag that shows the current year as a four-digit number.	See <i>Built-In Functions</i> in the InTouch® HMI Scripting and Logic Guide.

## Chapter 4

# Using Tag Dotfields to View or Change Tag Properties

## About Using Tag Dotfields to View or Change Tag Properties

Every type of InTouch tag has a unique set of properties that describe the data or possible conditions associated with the tag. A dotfield identifies a tag property. There is a dotfield for almost every tag property shown in the Tagname Dictionary. Some dotfields are common to every type of InTouch tag. For example, the .Name dotfield is always associated with a tag's name. Other dotfields apply only to the unique properties of a specific type of tag.

Using dotfields in a script, expression, or user input, you can monitor and modify a tag's properties while an application is running. The following example shows the syntax of a dotfield to access tag properties within a script or expression.

*tag\_name.property\_dotfield*

For example, to enable operators to change the HiHi alarm limit while an application is running, you can create an Analog - User Input touch link. Then, apply the link to a button that is defined with the Analog\_Tag.HiHiLimit dotfield expression. During run time, the operator simply clicks the button and types in a new value for the HiHi alarm limit for the tag.

You can use dotfields to allow input and output of data associated with a tag and you can use historical dotfields to modify the historical trend currently shown from a running application. For example, you can use dotfields in scripts that allow operators to modify historical trend scrolling, lock or reposition the scooters on the trend, or reassign the pens to new tags.

## Available Dotfields for Tag Types

Each type of InTouch tag has a set of dotfields associated with its unique properties. The following table shows an alphabetical list of dotfields for all types of tags.

Dotfields	Tag Types			
	Memory	I/O	Indirect	Miscellaneous

	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
.Ack	•	•	•		•	•	•		•	•		•			
.AckDev		•	•			•	•			•		•			
.AckDsc	•				•				•			•			
.AckROC	•	•	•			•	•			•		•			
.AckValue		•	•			•	•			•		•			
.Alarm	•	•	•		•	•	•		•	•		•			
.AlarmAccess														•	
.AlarmAckModel	•	•	•		•	•	•		•	•					
.AlarmClass														•	
.AlarmComment	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
.AlarmDate														•	
.AlarmDev		•	•			•	•			•		•			
.AlarmDevCount		•	•			•	•			•		•			
.AlarmDevDeadband		•	•			•				•					
.AlarmDevUnAckCount		•	•			•	•			•		•			
.AlarmDisabled	•	•	•		•	•	•		•	•		•			
.AlarmDsc	•				•				•			•			
.AlarmDscCount	•				•				•			•			
.AlarmDscDisabled	•				•				•			•			
.AlarmDscEnabled	•				•				•			•			
.AlarmDscInhibitor	•				•				•			•			
.AlarmDscUnAckCount	•				•				•			•			
.AlarmEnabled	•	•	•		•	•	•		•	•		•			
.AlarmGroup														•	
.AlarmGroupSel														•	

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
<b>.AlarmHiDisabled</b>		•	•			•	•			•					
<b>.AlarmHiEnabled</b>		•	•			•	•			•					
<b>.AlarmHiHiDisabled</b>		•	•			•	•			•					
<b>.AlarmHiHiEnabled</b>		•	•			•	•			•					
<b>.AlarmHiHiInhibitor</b>		•	•			•	•			•					
<b>.AlarmHiInhibitor</b>		•	•			•	•			•					
<b>.AlarmLimit</b>														•	
<b>.AlarmLoDisabled</b>		•	•			•	•			•					
<b>.AlarmLoEnabled</b>		•	•			•	•			•					
<b>.AlarmLoInhibitor</b>		•	•			•	•								
<b>.AlarmLoLoDisabled</b>		•	•			•	•			•					
<b>.AlarmLoLoEnabled</b>		•	•			•	•			•					
<b>.AlarmLoLoInhibitor</b>		•	•			•	•			•					
<b>.AlarmMajDevDisabled</b>		•	•			•	•			•					
<b>.AlarmMajDevEnabled</b>		•	•			•	•			•					
<b>.AlarmMajDevInhibitor</b>		•	•			•	•			•					
<b>.AlarmMinDevDisabled</b>		•	•			•	•			•					
<b>.AlarmMinDevEnabled</b>		•	•			•	•			•					
<b>.AlarmMinDevInhibitor</b>		•	•			•	•			•					
<b>.AlarmName</b>														•	
<b>.AlarmOprName</b>														•	
<b>.AlarmOprNode</b>														•	



Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
.AlarmPri														•	
.AlarmProv														•	
.AlarmROC		•	•			•	•			•					
.AlarmROCCount		•	•			•	•			•					
.AlarmROCDisabled		•	•			•	•			•					
.AlarmROCEnabled		•	•			•	•			•					
.AlarmROCIhibitor		•	•			•	•			•					
.AlarmROCUAckCount		•	•			•	•			•					
.AlarmState														•	
.AlarmTime														•	
.AlarmTotalCount	•	•	•		•	•	•		•	•		•			
.AlarmType														•	
.AlarmUnAckCount	•	•	•		•	•	•		•	•		•			
.AlarmUserDefNum1	•	•	•		•	•	•		•	•		•			
.AlarmUserDefNum1Set	•	•	•		•	•	•		•	•		•			
.AlarmUserDefNum2	•	•	•		•		•		•	•		•			
.AlarmUserDefNum2Set	•	•	•		•	•	•		•	•		•			
.AlarmUserDefStr	•	•	•		•	•	•		•	•		•			
.AlarmUserDefStrSet	•	•	•		•	•	•		•			•			
.AlarmValDeadband		•	•			•	•			•					
.AlarmValue			•				•			•		•		•	
.AlarmValueCount		•	•			•	•			•		•			
.AlarmValueUnAckCount		•	•			•	•			•		•			

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
.Caption														•	
.ChartLength													•		
.ChartStart													•		
.Comment	•	•	•	•	•	•	•	•	•	•	•	•	•		•
.DevTarget		•	•			•	•			•					
.DisplayMode													•		
.Enabled														•	
.EngUnits		•	•			•	•			•					
.Freeze														•	
.HiHiLimit		•	•			•	•			•					
.HiHiSet		•	•			•	•			•					
.HiHiStatus		•	•			•	•			•					
.HiLimit		•	•			•	•			•					
.HiSet		•	•			•	•			•					
.HiStatus		•	•			•	•			•					
.ListChanged														•	
.ListCount														•	
.ListIndex														•	
.LoLimit		•	•			•	•			•					
.LoLoLimit		•	•			•	•			•					
.LoLoSet		•	•			•	•			•					
.LoLoStatus		•	•			•	•			•					
.LoSet		•	•			•	•			•					

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
<b>.LoStatus</b>		•	•			•	•			•					
<b>.MajorDevPct</b>		•	•			•	•			•					
<b>.MajorDevSet</b>		•	•			•	•			•					
<b>.MajorDevStatus</b>		•	•			•	•			•					
<b>.MaxEU</b>		•	•			•	•			•					
<b>.MaxRange</b>												•			
<b>.MaxRaw</b>		•	•			•	•								
<b>.MinEU</b>		•	•			•	•			•					
<b>.MinorDevPct</b>		•	•			•	•			•					
<b>.MinorDevSet</b>		•	•			•	•			•					
<b>.MinorDevStatus</b>		•	•			•	•			•					
<b>.MinRange</b>												•			
<b>.MinRaw</b>		•	•			•	•			•					
<b>.Name</b>	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
<b>.NewIndex</b>														•	
<b>.NextPage</b>														•	
<b>.Normal</b>	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
<b>.NumAlarms</b>														•	
<b>.OffMsg</b>	•			•				•							
<b>.OnMsg</b>	•			•				•							
<b>.PageNum</b>														•	
<b>.Pen1 through .Pen8</b>												•			

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
<b>.PendingUpdates</b>														•	
<b>.PrevPage</b>														•	
<b>.PriFrom</b>														•	
<b>.PriTo</b>														•	
<b>.Quality</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualityLimit</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualityLimitString</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualityStatus</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualityStatusString</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualitySubstatus</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QualitySubstatusString</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.QueryState</b>														•	
<b>.QueryType</b>														•	
<b>.RawValue</b>	•	•	•		•	•	•		•	•		•			
<b>.ReadOnly</b>														•	
<b>.Reference</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.ReferenceComplete</b>	•	•	•	•	•	•	•	•	•	•	•				
<b>.ROCPct</b>		•	•			•	•			•					
<b>.ROCSet</b>		•	•			•	•								
<b>.ROCStatus</b>		•	•			•	•			•					
<b>.ScooterLockLeft</b>													•		
<b>.ScooterLockRight</b>													•		
<b>.ScooterPosLeft</b>													•		

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
.ScooterPosRight													•		
.Successful														•	
.SuppressRetain														•	
.TagID	•	•	•		•	•	•		•	•					
.TimeDate	•	•	•	•	•	•	•	•	•	•	•				
.TimeDateString	•	•	•	•	•	•	•	•	•	•	•				
.TimeDateTime	•	•	•	•	•	•	•	•	•	•	•				
.TimeDay	•	•	•	•	•	•	•	•	•	•	•				
.TimeHour	•	•	•	•	•	•	•	•	•	•	•				
.TimeMinute	•	•	•	•	•	•	•	•	•	•	•				
.TimeMonth	•	•	•	•	•	•	•	•	•	•	•				
.TimeMsec	•	•	•	•	•	•	•	•	•	•	•				
.TimeSecond	•	•	•	•	•	•	•	•	•	•	•				
.TimeTime	•	•	•	•	•	•	•	•	•	•	•				
.TimeTimeString	•	•	•	•	•	•	•	•	•	•	•				
.TimeYear	•	•	•	•	•	•	•	•	•	•	•				
.TopIndex														•	
.TotalPages														•	
.UnAck	•	•	•		•	•	•		•	•		•			
.UpdateCount													•		
.UpdateInProgress													•		
.UpdateTrend													•		
.Value(Tagname)	•	•	•	•	•	•	•	•	•	•	•				•

Dotfields	Tag Types														
	Memory				I/O				Indirect			Miscellaneous			
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Alarm Group	Hist Trend	Distributed Alarm Object / Controls	TagID
.Value(Windows Control)														•	
.Visible														•	

Dotfields can be categorized by their intended function. For more information about dotfield functional categories, see the following sections:

Category	See
Values and limits	<i>Changing the Value Limits of a Tag</i> on page 46.
Alarm parameters	<i>Controlling Alarm Properties of Tags and Groups at Run Time</i> in the InTouch® HMI Alarms and Events Guide.
I/O	<i>Data Access with I/O</i> on page 56.
Distributed Alarm Object	<i>Appendix A, Working with the Distributed Alarm Display Object</i> , in the InTouch® HMI Alarms and Events Guide.
Trend display	<i>Trending Tag Data</i> on page 151.
Window controls	<i>Wizards</i> in the InTouch® HMI Visualization Guide.

## Changing the Value Limits of a Tag

Raw input data from an I/O Server is transformed to a range of values appropriate for an InTouch application. The values of a tag are clamped to a range bounded by minimum and maximum values specified by the **Min Raw** and **Max Raw** properties of the Tagname Dictionary.

Then, these raw values are converted to a range of engineering units set by the **Min EU** and **Max EU** options. You can use a set of dotfields to monitor and modify a tag’s raw value and engineering units ranges.

The following table lists the dotfields that monitor or change tag values while an application is running.

Dotfield	Read/Write	Shows
<b>.MinRaw</b>	Read-only	The low clamp setting of a raw value received from an I/O Server.
<b>.MaxRaw</b>	Read-only	The high clamp setting of a raw value received from an I/O Server.
<b>.MinEU</b>	Read-only	The minimum value in engineering units assigned to the tag.
<b>.MaxEU</b>	Read-only	The maximum value in engineering units assigned to the tag.
<b>.EngUnits</b>	Read/Write	The text value assigned to an analog tag from the <b>Eng Units</b> option of the Tagname Dictionary.
<b>.RawValue</b>	Read-only	The actual discrete or analog value received by the tag from an I/O Server before scaling is applied.
<b>.Value</b>	Read/Write	The current value of a tag.
<b>.OnMsg</b>	Read/Write	The message assigned to a discrete tag when its value evaluates to True, On, or 1.
<b>.OffMsg</b>	Read/Write	The message assigned to a discrete tag when its value evaluates to False, Off, or 0.
<b>.Comment</b>	Read/Write	The tag comment specified from the Tagname Dictionary.

## Viewing the Raw Value Limit

Raw input tag values from an I/O Server may require clamping before they can be used in an InTouch application. Clamping restricts raw values to a range with defined lower and upper limits. The **.MinRaw** and **.MaxRaw** dotfields show the lower and upper boundaries of the raw input range.

### .MinRaw Dotfield

The **.MinRaw** dotfield shows the Min Raw low clamp setting assigned to a tag. The value for **.MinRaw** dotfield comes from the **Min Raw** value assigned to the I/O tag in the Tagname Dictionary. Any raw value less than this setting is clamped to this minimum value.

#### Category

Tags

### Usage

```
tag_name.MinRaw;
```

### Parameter

*Tag\_name*

The name of any I/O integer, I/O real, and indirect analog tag.

### Remarks

This read-only dotfield shows the value assigned to the Min Raw low clamp setting.

### Data Type

Real or integer (read-only).

### Valid Values

Any analog value.

### Example

The following script shows an error window if the raw value associated with pump inlet pressure is outside of the lower and upper value boundaries set by the tag's **Min Raw** and **Max Raw** properties.

```
IF ((PumpInP.RawValue > PumpInP.MaxRaw) OR
    (PumpInP.RawValue < PumpInP.MinRaw)) THEN
    Show "Instrument Failure Window";
ENDIF;
```

### See Also

**.EngUnits, .MinEU, .MaxEU, .MaxRaw, .RawValue**

## .MaxRaw Dotfield

The **.MaxRaw** dotfield shows the Max Raw high clamp setting assigned to an I/O tag from the **Max Raw** property in the Tagname Dictionary. Any raw data value that exceeds this setting is clamped to this maximum raw value.

### Category

Tags

### Usage

```
Tag_name.MaxRaw
```

### Parameter

*Tag\_name*

The name of any I/O integer, I/O real, and indirect analog tag.

### Remarks

This read-only dotfield shows the value assigned to the Max Raw high clamp setting.

### Data Type

Real or integer (read-only).



## Valid Values

Any analog value.

## Example

This script determines if a tag value is out of normal operating range and a window is shown if this is the case.

```
IF ((Temp01.RawValue > Temp01.MaxRaw) OR (Temp01.RawValue <
    Temp01.MinRaw))THEN
    Show "Instrument Failure Window";
ENDIF;
```

## See Also

**.EngUnits, .MinEU, .MaxEU, .MinRaw, .RawValue**

## Viewing the Raw Value of a Tag

The **.RawValue** dotfield shows the actual discrete or analog value of a monitored property received from an I/O Server. The raw value is the actual input value before clamping and scaling are applied to normalize the value to the tag's engineering units.

## .RawValue Dotfield

The **.RawValue** dotfield shows the actual value received from an I/O Server by WindowViewer. The **.RawValue** dotfield allows you to access the value of an I/O tag before InTouch applies scaling.

## Category

Tags

## Usage

```
Tag_name.RawValue
```

## Parameter

*Tag\_name*

The name of any I/O integer, I/O real, I/O discrete, indirect discrete, and indirect analog tag.

## Remarks

This read-only dotfield is used to show the actual discrete or analog I/O value before InTouch applies scaling.

## Data Type

Any data appropriate for the type of tag associated with the **.RawValue** dotfield. For example, real numbers for real tags or discrete values for discrete tags (read-only).

## Example

The following script issues a warning message when the raw pump inlet pressure is below or above the tag's minimum and maximum clamping limits.

```
IF ((PumpInP.RawValue > PumpInP.MaxRaw) OR (PumpInP.RawValue < PumpInP.MinRaw)) THEN
    AlarmMessage = "Pump sensor is out of calibration or requires replacement.";
ENDIF;
```

## See Also

**.EngUnits, .MinEU, .MaxEU, .MinRaw, .MaxRaw**

## Viewing the Engineering Units Value Limit

A value from an I/O Server is considered raw when it first arrives in WindowViewer. Raw values may require scaling. The InTouch HMI performs an arithmetic transformation on the raw clamped input values to scale them to the engineering units range of the tag. I/O Integer and real tag types include **Min EU** and **Max EU** properties that show the lower and upper boundaries of the engineering unit range.

### .MaxEU Dotfield

The **.MaxEU** dotfield shows the maximum engineering unit value assigned to the specified tag from the Tagname Dictionary.

#### Category

Tags

#### Usage

```
Tag_name.MaxEU
```

#### Parameter

*Tag\_name*

Any integer, real, or indirect analog tag.

#### Remarks

The **.MaxEU** dotfield is used to scale raw data values to an engineering unit range defined for the tag. It defines the upper limit of engineering unit range.

#### Data Type

Real for real tags and integer for integer tags (read-only).

#### Valid Values

Depends on the type of tag specified.

#### Example

A level gauge is read by a Programmable Logic Controller in the field. The level transmitter sends a signal that ranges between 4 and 20mA. The PLC converts this signal to an integer value between 0 and 4095. This value is assigned to the TankTwoLevel tag.

Displaying the raw value (between 0 and 4095) provides no useful data to the operator. It is necessary to scale this value to an appropriate engineering range.

To accomplish this, the Minimum engineering unit and Maximum engineering unit fields must be set up correctly. In our example, if the raw value of 0 (4mA from the field) translated to "0 Gallons" and the value of 4095 (20mA from the field) translated to "100 Gallons", the following set up would be required to show the correct value on the screen:

```
TankTwoLevel.MinRaw = 0;  
TankTwoLevel.MaxRaw = 4095;  
TankTwoLevel.MinEU = 0;
```

```
TankTwoLevel.MaxEU = 100;
```

With these settings, when the raw value in the field is 4095, the value shown to the operator is 100.

**See Also**

**.EngUnits, .MinEU, .MinRaw, .MaxRaw, .RawValue**

**.MinEU Dotfield**

The **.MinEU** dotfield shows the minimum engineering unit value assigned to the specified tag from the Tagname Dictionary.

**Category**

Tags

**Usage**

```
Tag_name.MinEU
```

**Parameter**

*Tag\_name*

Any integer, real, or indirect analog tag.

**Remarks**

The **.MinEU** dotfield is used to scale raw data values to an engineering unit range defined for the tag. It defines the lower limit of engineering unit range.

**Data Type**

Real for real tags and integer for integer tags (read-only).

**Valid Values**

Depends on the type of tag specified.

**Example**

This example assigns the engineering unit range defined for the Tag1 tag to the **AbsoluteTagRange** tag.

```
AbsoluteTagRange = (Tag1.MaxEU - Tag1.MinEU);
```

**See Also**

**.EngUnits, .MaxEU, .MinRaw, .MaxRaw, .RawValue**

## Changing the Engineering Units of a Tag

You can associate a dotfield to a tag to determine the text value of the engineering unit assigned to the tag.

**.EngUnits Dotfield**

The **.EngUnits** dotfield shows the text value assigned to an analog tag with the **Eng Units** property. The **.EngUnits** dotfield shows the engineering unit as a text value.

---

**Note:** Values written to this field are not retentive.

---

**Category**

Tags

**Usage**

*Tag\_name*.EngUnits

**Parameter**

*Tag\_name*

Any integer, real, or indirect analog tag.

**Data Type**

Message (read/write).

**Remarks**

The **.EngUnits** dotfield does not affect the scale, conversion, or format of the actual data associated with the tag.

**Valid Values**

Any string containing 0 to 31 characters.

**Example**

The following script calls a Fahrenheit temperature conversion function if the tag’s engineering unit is Celsius.

```
IF Temperature.EngUnits == "Celsius" THEN
    CALL TempFConvert(Temperature);
ENDIF;
```

**See Also**

**.MinEU, .MaxEU, .MinRaw, .MaxRaw, .RawValue**

## Viewing the Value of a Tag in Engineering Units

InTouch tags are assigned a default dotfield when no dotfield is explicitly specified within the application.

### .Value Dotfield

The **.Value** dotfield shows the current value of the specified tag in engineering units. **.Value** is the default InTouch dotfield implicitly applied to all tags. If a tag does not have an assigned dotfield, the **.Value** dotfield is assumed.

**Category**

Tags

**Usage**

*tag\_name*.Value

**Parameter**

*tag\_name*

Any type of tag except the Hist Trend tag.

### Remarks

You rarely need to use the **.Value** dotfield. However, in some instances, it makes a calculation or parameter usage more clear.

### Data Type

The same as the specified tag's type (read/write).

### Valid Values

Depends on the type of tag specified.

### Example

The following statement sets the value of the memory integer **PumpRPM** tag equal to 100:

```
PumpRPM.Value=100;
```

which is functionally equivalent to:

```
PumpRPM=100;
```

## Viewing or Changing Discrete Tag Messages

The **.OnMsg** and **.OffMsg** dotfields show the messages assigned to a discrete tag's on or off states from the Tagname Dictionary. A discrete tag's on and off messages are short strings with a maximum of 15 characters.

### .OnMsg Dotfield

The **.OnMsg** dotfield allows you to access the On message assigned to a discrete tag from the Tagname Dictionary.

#### Category

Tags

#### Usage

```
Tag_name.OnMsg
```

#### Parameter

*Tag\_name*  
 Any discrete tag.

#### Data Type

Message (read/write). Values written to this dotfield are not retentive.

#### Valid Values

Any string containing 0 to 15 characters.

#### Example

The following statement issues a message if the indirect **IndPumpState** tag On message is assigned a string value of "Pump1 running".

```
IF IndPumpState.OnMsg == "Pump1 running" THEN
  TypeOfTag = "The IndPumpState tag is assigned to Pump1.";
ENDIF;
```

## See Also

[.OffMsg](#)

## .OffMsg Dotfield

The **.OffMsg** dotfield allows you to access the Off message assigned to a discrete tag from the Tagname Dictionary.

### Category

Tags

### Usage

```
Tag_name.OffMsg
```

### Parameter

*Tag\_name*  
Any discrete tag.

### Data Type

Message (read/write). Values written to this dotfield are not retentive.

### Valid Values

Any string containing 0 to 15 characters.

### Example

The following statement assigns the appropriate string to the **StateMessage** tag according to the state of the **MyDiscrete** tag.

```
StateMessage=Dtext (MyDiscrete, MyDiscrete.OnMsg, MyDiscrete.OffMsg);
```

## See Also

[.OnMsg](#)

## Viewing or Changing the Comment of a Tag

A tag comment can be permanently changed only from the Tagname Dictionary. You can assign another comment with the **.Comment** dotfield while the application is running. This only changes the comment for the duration of the run-time session. It does not permanently change the tag's comment in the Tagname Dictionary. After WindowViewer is shut down and restarted, the original comment is assigned to the tag.

## .Comment Dotfield

The **.Comment** dotfield shows the comment assigned to a tag from the Tagname Dictionary. A tag comment can be a string up to 50 characters.

### Category

Tags

### Usage

```
Tag_name.Comment
```

**Parameter**

*Tag\_name*  
Any tag name.

**Remarks**

The comment associated with a tag can be modified with the .Comment dotfield while an InTouch application is running. After the application is stopped, the original comment specified from the Tagname Dictionary remains assigned to the tag.

**Data Type**

Message

**Valid Values**

Any string from 1 to 50 characters.

**Example**

The following statement creates an operator message by combining a tag's assigned comment to the name of the tag:

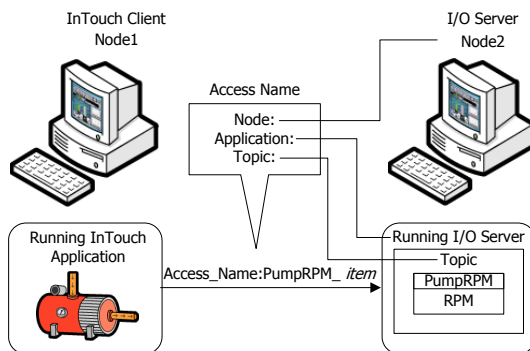
```
OperatorMessage=PumpRPM.Name + " has a comment of: " + PumpRPM.Comment;
```

## Chapter 5

# Data Access with I/O

## About Data Access with I/O

You can develop distributed applications in which the functional components of an InTouch system are located on different nodes. The figure below shows how you configure an I/O request for data stored on another node.



You can set up an InTouch application to identify an element of data stored on another node by using a three-part addressing convention. This convention includes the node, application, and topic names. To obtain data from a remote node, you need to configure an Access Name for your InTouch application that specifies these three items.

For example, if you want to access data from a remote I/O Server running on another node, your Access Name consists of the following:

Access Name Option	Description
Node Name	Node name of the computer running the I/O Server program.
Application Name	Name of the I/O Server program running on the node.  For example, "UA_SampleServer" might be the name of an OPC UA server.  For more information about the application names associated with Operations Integration (OI) servers, refer to the OI server documentation.
Topic Name	Label assigned to the I/O Server Device Group.



If you use an Excel spreadsheet as your InTouch data source, you can define your Access Name as follows:

---

Access Name Option	Description
Node Name	Node name of the computer running the Excel program.
Application Name	Excel is the Application Name.
Topic Name	Name of the Excel book and spreadsheet containing the requested data. For example, [Book1]Sheet1.

In addition to the node, application, topic, and item, you need to specify the type of data located on the remote node. This information determines the I/O type for the tag when it is defined in the Tagname Dictionary.

## Working with OI Gateway Communication Driver

OI Gateway Communication Driver is an application that acts as a portal and protocol converter to allow two computer systems or programs to communicate with one another. You can use OI Gateway to link clients and data sources that communicate using different protocols, such as OPC and OPC UA. You can access OI Gateway from InTouch to connect to field devices.

InTouch leverages OI Gateway to use OPC, OPC UA, and other protocols to communicate between automation and control applications, field systems and devices, and business and office applications.

OI Gateway is bundled with WindowViewer and is automatically installed when you install InTouch. OI Gateway can also be installed as a standalone application.

## About the Gateway Communication Driver

The OI Gateway acts as a communications protocol converter. OI Gateway can be used to link clients and data sources that communicate using different protocols.

This user assistance publication covers only the information you need to configure and run the OI Gateway component. The documentation that accompanies the related components provide details on their operation.

You can troubleshoot problems with OI Gateway using the ArchestrA Log Viewer, a snap-in to the System Management Console (SMC). See the Log Viewer help file to find information on:

- Viewing error messages.
- Determining which messages are shown.
- Bookmarking error messages.

You may also be able to troubleshoot problems using your client application, such as the InTouch HMI software. The client application can use system device items to determine the status of nodes and the values of some parameters.

The basic rules for OI Gateway include:

- One instance of OI Gateway can run per node.
- The Gateway can be activated and deactivated using the OI Server Manager snap-in.

- The Gateway can be activated as a COM Server (OPC Server) using standard COM activation mechanisms.
- The Gateway can be run only out-of-proc within OPC clients.
- The Gateway can communicate only with ArcestrA data source components delivered with Application Server v2.0 and later. Earlier versions of Application Server are not supported.

## Getting Started Workflow

The following steps explain the workflow of getting started with OI Gateway:

1. Install InTouch.
  - OI Gateway is included as part of the InTouch installation.
  - You must separately install OPC and OPC UA servers.
2. Install OPC or OPC UA servers and configure them to communicate with your field devices.
3. Configure other protocols as necessary.
4. Create your application in InTouch.
5. Configure OI Gateway to point to your installed OPC server or OPC UA server or other data sources.
6. Activate OI Gateway.

For more information about setting up OI Gateway, see *Setting up the OI Gateway for the First Time* in the OI Gateway Communications Driver Help.

## Supported InTouch Communication Protocols

You can configure the InTouch HMI to use DDE or SuiteLink. The InTouch HMI also supports the ArcestrA Message Exchange communication protocol.

For more information about using Message Exchange within an InTouch application, see *Accessing Application Server Data from InTouch* on page 78.

## Dynamic Data Exchange

The Dynamic Data Exchange (DDE) communication protocol enables Windows applications to communicate with each other. DDE implements a client-server relationship between two concurrently running applications. The server application provides data and accepts requests from any other application interested in its data. Requesting applications are called clients. An InTouch application can be simultaneously both a client and a server.

## SuiteLink

SuiteLink is a TCP/IP-based protocol designed specifically for industrial applications. SuiteLink provides data integrity, high throughput, and simple diagnostic procedures. The SuiteLink protocol is supported by Microsoft Windows NT 4.0 and later.

SuiteLink is not a replacement for DDE or NetDDE. Each connection between a client and a server depends on your network requirements.

SuiteLink provides the following:

- Value Time Quality (VTQ) places a time stamp and quality indicator on all data values delivered to VTQ-aware clients.
- Extensive diagnostics of data throughput, server loading, computer resource consumption, and network transport are made accessible through the Microsoft Windows operating system performance monitor.
- Consistent high data volumes can be maintained between applications even if the applications are on a single node or distributed over a large set of nodes.
- The network transport protocol is TCP/IP using Microsoft's standard Winsock interface.
- You can securely configure the SuiteLink node using the TLS 1.2 protocol. For more information, refer to the OI Core Communications Driver Help.

## Troubleshooting SuiteLink Communication Problems

If you encounter SuiteLink communication problems, do the following:

- Confirm that Microsoft TCP/IP is operational on the computer on which the InTouch HMI is installed.
- Verify the computer node name is 15 characters or fewer.
- Confirm that SuiteLink is running as a service on the computer on which the InTouch HMI is installed.
- SuiteLink is automatically installed during the InTouch installation. The SuiteLink service starts automatically. If the SuiteLink service stops, you must start it again.

## Accessing the Server as a Standard User

For InTouch tag-based applications in a Tag Server architecture, our recommendation to achieve a more secure configuration with encrypted SuiteLink communications is to run InTouch WindowViewer as a service at the tag server node. This allows WindowViewer to run under a non-interactive user acting as a SuiteLink Server with the necessary user privileges to access the private keys with which SuiteLink Server – Client communications are encrypted, something not possible when using an interactive user.

This is necessary because the user under which WindowViewer is running provides the security context used to retrieve the private keys required by the SuiteLink server for secure communications. If WindowViewer is run as an interactive application, then the user is an interactive user which, as described previously, cannot access the private keys for secure encrypted SuiteLink communications.

Running InTouch as a tag server as a service can be coupled with InTouch Tag Server Client licenses at the client nodes, which limit InTouch to communicate with a tag server only.

Users working in this scenario commonly utilize PC virtualization technologies like HyperV or VMWare as examples, and run WindowViewer as a service within that VM, to reduce the Total Cost of Ownership of their solution.

## OPC

OPC is not itself a communication protocol, but functions as a driver—a non-proprietary set of standard interfaces, based on Microsoft's OLE/COM technology. This standard makes possible interoperability between automation/control applications, field systems/devices, and business/office applications.

Avoiding the traditional requirement of software application developers to write custom drivers to exchange data with field devices, OPC defines a common, high-performance interface that permits this work to be done once, and then easily reused by HMI, SCADA, control and custom applications.

Over a network, OPC uses DCOM (Distributed COM) for remote communications.

## OPC UA

OPC Unified Architecture (OPC UA) is an industrial machine-to-machine communication protocol for interoperability. It provides process control with enhanced security, advanced communication, security, and information models, and cross-platform connectivity.

OPC UA is implemented as a client in OI Gateway.

OPC UA differs significantly from OPC. The following provides the key differences between classic OPC and OPC UA.

Classic OPC	OPC UA
Uses the COM/DCOM technology of Microsoft to communicate. It does not have configurable time-outs. It depends on the DCOM time-out, which is configured in the system.	Uses a services architecture to export data, which improves the ease of communication and connectivity.
Is dependent on Windows operating systems.	Is platform independent and can connect to a wide variety of devices and platforms.
Has limited security.	Has built-in security.
No built-in capabilities to handle problems, such as lost messages.	Has built-in capabilities to handle problems, such as lost messages.

## MQTT

MQTT, formerly called Message Queuing Telemetry Transport, is a publish/subscribe messaging protocol for use over TCP/IP. MQTT is designed to ensure that devices can communicate with each other while minimizing power and bandwidth requirements. It is a simple messaging protocol that is well-suited for use with devices that rely on slow or unreliable networks.

The MQTT protocol is an application layer specification, and has been published as standard ISO/IEC PRF 20922. MQTT uses a Publish-Subscribe mechanism which requires a mediating broker. The publishers send data to the broker, and subscribing clients receive data published to the broker. Only clients that have subscribed to a particular topic receive messages about that topic. The protocol supports bidirectional communication such that a device that is a publisher can also receive updates.

## Setting Up Access Names

You must associate InTouch I/O tags or remote tag references with an Access Name. An Access Name defines a communication link with another I/O data source. Each Access Name specifies an I/O address consisting of a node name, an application name, and a topic.

In a distributed application, I/O references can be set as global addresses to a network I/O Server, or local addresses to a local I/O Server.

InTouchView shows the visual interface of HMI applications designed specifically for use in an ArchestrA Application Server environment. InTouchView applications run as a client with Application Server acting as a server that provides most HMI functionality.

InTouchView applications offer only some of the standard functions available from full-featured InTouch applications. InTouchView applications cannot connect to I/O sources other than the ArchestrA Application Server Galaxy. You can only use the default Galaxy Access Name with your InTouchView application and cannot create Access Names.

### To create an Access Name

1. On the **Special** menu, click **Access Names**. The **Access Names** dialog box appears.
2. Click **Add**. The **Add Access Name** dialog box appears.

3. Set the properties of the **Add Access Name** dialog box. Do the following:
  - In the **Access** box, type a name that identifies this Access Name.
  - If the data resides on a network I/O Server, type the remote server's node name in the **Node Name** box.
  - In the **Application Name** box, type the actual program name of the I/O Server program from which data will be acquired.

If the I/O data source is a DAServer, type the name of the DAServer program, do not include the .exe file name extension of the program.

- In the **Topic Name** box, type the topic name you want to access.  
The topic name is an application-specific sub-group of data elements. In the case of data coming from a DAServer program, the topic name is the same name configured for the topic in the DAServer program. When communicating with Microsoft Excel, the topic name must be the name assigned to the book and spreadsheet when it was saved. For example, [Book1]Sheet1.
- Select the communication protocol to communicate with the I/O Server.
- Select the option to poll information stored on the server.

Option	Definition
<b>Advise all items</b>	Polls for all data whether or not it is in visible windows, alarmed, logged, trended, or used in a script. Selecting this option affects performance, so its use is not recommended.
<b>Advise only active items</b>	Polls for data shown in visible windows and data that is alarmed, logged, trended, or used in any script. <hr/> A button action script is not polled unless it appears in a visible window.

4. Select **Enable Secondary Source** if you want to select a secondary back up server. Otherwise, go to step 5.

- If you select the **Enable Secondary Source** option, the **Add Access Name** dialog box expands to show the configuration fields for the second source.
- Complete the options.
- Click **OK** after assigning all values to the secondary source back up server.
- When you are done specifying the Access Name, click **OK**. The **Access Names** dialog box reappears and shows the new Access Name is added to the list.

5. Click **Close**.

For more information about setting up your secondary server for failover switching, see *Using Failover Functionality with Access Names* on page 104.

## Deleting Access Names

You can delete an Access Name that you no longer need. Before you delete an Access Name, make sure of the following:

- No tags are associated with the Access Name.

- WindowViewer is stopped.

**To delete an Access Name**

1. On the **Special** menu, click **Access Names**. The **Access Names** dialog box appears. The current Access Names are listed.
2. To delete an Access Name, select it from the list and click **Delete**. A message appears requesting confirmation that the Access Name should be deleted.
3. Click **Yes**.
4. Click **Close** or repeat this procedure if you need to delete other defined Access Names.

## Accessing I/O Data with I/O Tags

The InTouch HMI can send and receive data from local or remote Windows applications with I/O tags. Each I/O type tag references a valid item in the I/O Server program. You can define different types of I/O tags in the Tagname Dictionary.

## Configuring I/O Tag Properties

You define the different types of I/O tags in the Tagname Dictionary.

### Specifying a Discrete I/O Tag

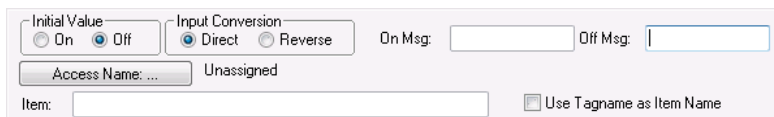
I/O discrete tags indicate the binary state of all inputs and outputs from programmable controllers, process computers, and data from network nodes.

An I/O discrete tag must be assigned an initial value of On or Off. You can also configure the discrete I/O tag to toggle to the opposite value of its binary source. You can specify messages that appear in the alarm event window when the process associated with the tag goes into an alarm state.

For more information about the overall procedure to create a tag from the Tagname Dictionary, see *Creating New Tags* on page 22.

**To define an I/O discrete tag**

1. Open the Tagname Dictionary and assign a name for a new tag.
2. Assign the tag as an I/O discrete type from the **Tag Types** dialog box. The detail portion of the **Tagname Dictionary** dialog box appears.



3. Do the following:
  - Select **On** or **Off** as the initial value associated with the tag.  
The InTouch HMI assigns this value to the tag when the application starts, but does not write this initial value to the I/O device.
  - Select **Direct** or **Reverse** as the input conversion applied to the value received from a remote I/O tag.

Input Conversion	Description
<b>Direct</b>	The input value is read unchanged directly from the I/O Server program.
<b>Reverse</b>	The I/O discrete value is toggled when read from the server program. For example, if the I/O input is 0, the value is automatically set to 1.

- Enter messages in the **On Msg** and **Off Msg** boxes that appear when the tag transitions in and out of an alarm state.

These messages are available for use in any animation link or script, regardless of whether the tag has alarms configured or not.

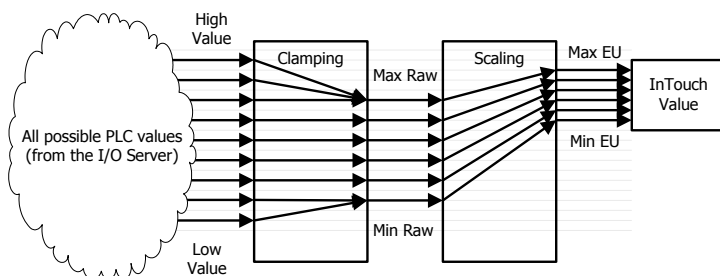
- If you define a discrete alarm that is active when the tag value is equal to 1 (On, True), the message entered in the **On Msg** box appears in the **Value** and **Limit** columns of your ActiveX alarm displays. When the tag’s alarm state returns to normal, the message entered in the **Off Msg** box appears in the **Value** column and the On message remains in the **Limit** column.
- If you define a discrete alarm that is active when the tag value is equal to 0 (Off, False), the message entered in the **Off Msg** box appears in the **Value** and **Limit** columns of your ActiveX alarm displays. When the tag’s alarm state returns to normal, the message entered in the **On Msg** box appears in the **Value** column and the Off message remains in the **Limit** column.

- Save your changes to the tag.

## Specifying Integer and Real I/O Tags

You must assign I/O integer and real tags a set of attributes that characterize numerical data sent between the InTouch application and external processes.

The InTouch HMI normalizes raw input data from a PLC. The figure below shows the process of clamping raw I/O data values and then scaling them to engineering units that can be shown from an InTouch application.



I/O integer and real tags include attributes that set minimum and maximum limits for raw input data sent by the PLC. The InTouch HMI clamps I/O values that are below or above the raw value range. Clamping reassigns values outside of the range to the minimum or maximum raw values.

I/O integer and real tags include attributes that scale clamped raw values within a range of engineering units. Minimum and maximum engineering unit attributes set the upper and lower boundaries of the scaled values.



When you define integer and real I/O tags, you specify the type of conversion to scale raw values when calculating engineering units. You can select Linear or Square Root.

For linear scaling, the result is calculated using linear interpolation between the minimum and maximum end points. The algorithm for linear scaling of input is:

$$EUValue = (RawValue - MinRaw) * ((MaxEU - MinEU) / (MaxRaw - MinRaw)) + MinEU$$

The algorithm for linear scaling of output is:

$$RawValue = (EUValue - MinEU) * ((MaxRaw - MinRaw) / (MaxEU - MinEU)) + MinRaw$$

For square root scaling, the minimum and maximum raw values are used for interpolation. This is useful for scaling inputs from nonlinear devices like pressure transducers. The algorithm for square root scaling of input is:

$$EUValue = \text{sqrt}(RawValue - MinRaw) * ((MaxEU - MinEU) / \text{sqrt}(MaxRaw - MinRaw)) + MinEU$$

The algorithm for square root scaling of output is:

$$RawValue = \text{square}((EUValue - MinEU) * (\text{sqrt}(MaxRaw - MinRaw) / (MaxEU - MinEU))) + MinRaw$$

### To define integer and real I/O tags

1. Open the Tagname Dictionary and assign a name for a new tag.
2. Assign I/O integer or I/O real as the type of tag from the **Tag Types** dialog box. The detail portion of the **Tagname Dictionary** dialog box appears.

The screenshot shows the 'Tagname Dictionary' dialog box with the following fields and values:

- Initial Value: 0
- Deadband: 15
- Eng Units: PSI
- Access Name: TankFarm1
- Item: PumpInP
- Min EU: 0
- Min Raw: 0
- Log Deadband: 0
- Max EU: 2500
- Max Raw: 45325
- Conversion:  Linear  Square Root
- Use Tagname as Item Name

3. Do the following:
  - In the **Initial Value** box, type the integer or real number associated with the tag when the application starts.  
The application does not write this initial value to the external process.
  - In the **Min EU** box, type the minimum engineering unit for the tag.
  - In the **Max EU** box, type the maximum engineering unit for the tag.
  - In the **Min Raw** box, type the minimum value of the low clamp for raw I/O integer or real numbers.
  - In the **Max Raw** box, type the maximum value of the high clamp for raw I/O integer or real numbers.
  - In the **Eng Units** box, type the label to use for the tag's engineering units.
  - Select **Linear** or **Square Root** as the type of conversion to scale raw values when calculating engineering units.
4. Save your changes to the tag.

## Specifying a Message I/O Tag

You can specify an I/O message tag options that specify the network address of remote processes. Its message properties are the same as a memory message tag.

### To define memory and I/O message tag values

1. Open the Tagname Dictionary and assign a name for a new tag.

2. Select **I/O Message** as the type of tag from the **Tag Types** dialog box. The detail portion of the **Tagname Dictionary** dialog box appears.

3. In the **Maximum Length** box, type the maximum number of characters allowed in the tag's message. You can enter messages to a maximum of 131 characters.
4. In the **Initial Value** box, type the text string that you want shown when WindowViewer starts the application.
5. Save your changes to the tag.

### Setting I/O Access Parameters

You can set the I/O attributes of tags in the Tagname Dictionary. These attributes identify the external data source associated with the tag.

These steps only explain how to specify I/O attributes from the Tagname Dictionary. For more information about configuring Galaxies and Access Names, see *Data Access with I/O* on page 56.

#### To set tag I/O attributes

1. Assign an I/O tag type to the tag from the **Tag Types** dialog box. The detail portion of the **Tagname Dictionary** dialog box appears.

2. Click **Access Name** to define or select the Access Name assigned to the tag. The **Access Names** dialog box appears showing a list of current Access Names recognized by the InTouch HMI. (Galaxy is the default Access Name for an ArcestrA connection.)

3. Add an Access Name or accept the default.
4. Select the data point in the server program that the I/O tag will read and write data.
  - o To read and write data to and from a process data point in the server program, type the Item Name in the **Item** box. For example, to read a value from a register in a PLC, type the identity of the register as the Item Name. For example:

To use the register 1 of an Allen-Bradley PLC as the Item Name, type R1 in the **Item** box.

To use the least significant bit of register 1 of an Allen-Bradley PLC as the Item Name, type R1:0 in the **Item** box.

- To use the tag as the item, select **Use Tagname as Item Name**.

## Retrieving Information About I/O Tags at Run Time

You can write scripts with functions that return the names of the node, application, and topic specified in an Access Name definition.

### IOGetNode() Function

The **IOGetNode()** function returns the node address defined for a specific Access Name to a tag associated with the function in the script.

#### Category

Miscellaneous

#### Syntax

```
IOGetNode("AccessName");
```

#### Argument

*AccessName*

The existing Access Name to return node information for.

#### Remarks

You can specify the Access Name as a literal string, or as a string value provided by other InTouch tags or functions.

#### Example

The following example returns the node information for the ModbusPLC1 Access Name to the **NodeName** tag.

```
NodeName = IOGetNode("ModbusPLC1");
```

### IOGetApplication() Function

The **IOGetApplication()** script function returns the application name defined for a specific Access Name to a tag assigned as an argument of the function.

#### Category

Miscellaneous

#### Syntax

```
IOGetApplication("AccessName");
```

#### Argument

*AccessName*

The existing Access Name in which the application is defined.

### Remarks

You can specify the Access Name as a literal string, or as a string value provided by other InTouch tags or functions.

### Example

The example returns the name of the application specified for the ModbusPLC1 Access Name to the **AppName** tag.

```
AppName = IOGetApplication ("ModbusPLC1");
```

## IOGetTopic() Function

The **IOGetTopic()** script function returns the topic name defined for a specific Access Name to a tag associated with the function in the script.

### Category

Miscellaneous

### Syntax

```
IOGetTopic("AccessName");
```

### Argument

*AccessName*

The Access Name whose topic name is returned.

### Remarks

The Access Name can be specified as a literal string, or it can be a string value provided by other InTouch message tags or functions.

### Example

This example returns topic information for the ModbusPLC1 Access Name to the **TopicName** tag.

```
TopicName = IOGetTopic("ModbusPLC1");
```

## Dynamically Changing I/O Tag References at Run Time

The InTouch HMI uses dynamic references to view data points whose values are only needed temporarily, such as in diagnostic applications. Using Dynamic Reference Addressing allows you to address multiple data sources with a single tag.

You can use several methods to dynamically reference multiple data sources with a single tag:

- Assign different Access Name characteristics with the **.Reference** dotfield of an I/O tag
- Use the **IOSetItem()** script function to set an I/O tag's **.Reference** dotfield
- Use the **IOSetAccessName()** script function to change the characteristics of an Access Name during run time

### .Reference Dotfield

You can implement Dynamic Reference Addressing by assigning a valid reference to the **.Reference** dotfield of an I/O tag. You can use the **.Reference** dotfield to dynamically change the data source by modifying the characteristics of the Access Name assigned to the I/O tag.

The syntax of the **.Reference** dotfield is:

<code>tag.Reference="accessname.item"</code>	Changes the Access Name and item assigned to a tag.
<code>tag.Reference=".[item]"</code>	Changes the item assigned to an I/O tag.
<code>tag.Reference="accessname."</code>	Changes the Access Name of an I/O tag.
<code>tag.Reference=""</code>	Deactivates the I/O tag.

Each I/O type tag has a **.ReferenceComplete** dotfield. The value of this discrete dotfield indicates if the item requested in the **.Reference** dotfield is reflected in the **.Value** dotfield.

The **.ReferenceComplete** field is set to false (0) when the application starts in WindowViewer. When it is confirmed that the **.Value** dotfield is being updated by the source specified in the **.Reference** dotfield, the **.ReferenceComplete** value is set to true (1). If the **.Reference** dotfield is changed, the **.ReferenceComplete** dotfield is automatically set to false (0), and then updated to true (1) when the new value is updated.

## IOSetItem() Function

You can implement Dynamic Reference Addressing by using the **IOSetItem()** function within a script. **IOSetItem()** includes arguments to change the values assigned to the **.Reference** dotfield of an I/O tag during run time.

### Category

Miscellaneous

### Syntax

```
IOSetItem ("Tag", "AccessName", "Item");
```

### Arguments

*Tag*

Any InTouch I/O tag enclosed in quotation marks.

*AccessName*

The Access Name assigned to the I/O tag.

*Item*

The Item assigned to the I/O tag.

The Tag, AccessName, and Item arguments can be specified as literal strings or they can be string values provided by other InTouch tags or functions.

### Examples

In the following example, the **.Reference** dotfield of the **PumpInP1** tag is changed to point to the excel Access Name and the R1C1 item.

```
IOSetItem("PumpInP1", "excel", "R1C1");
```

or

```
Number = 1;  
TagNameString = "PumpInP" + Text(Number, "#");
```

```
IOSetItem(TagNameString, "excel", "R1C1");
```

If an empty string ("") is specified for both the Access Name and item values, then the tag is deactivated. For example, the **PumpInP2** tag is deactivated by:

```
IOSetItem("PumpInP2", "", "");
```

If a null is specified only for an item, then the tag's current item value is retained and its Access Name value is updated. For example, the following changes the Access Name for the **PumpInP3** tag to excel2 without affecting its current Item value:

```
IOSetItem("PumpInP3", "excel2", "");
```

Likewise, if a null string is specified only for an Access Name, then the tag's current Access Name value is retained and its item value is updated. The following example changes the Item for the **PumpInP4** tag to R1C2 without affecting its current Access Name value:

```
IOSetItem("PumpInP4", "", "R1C2");
```

## IOSetAccessName() Function

You can implement Dynamic Reference Addressing by using the **IOSetAccessName()** function within a script. **IOSetAccessName()** modifies the application or topic name characteristics of an I/O tag's Access Name during run time.

---

**Note:** When the IOSetAccessName() function is processed, there is a time delay while the existing conversation is terminated and the new conversation is started. During this period, any attempted POKEs or writes to the new topic are lost.

---

### Category

Miscellaneous

### Syntax

```
IOSetAccessName("AccessName", "NodeName", "AppName", "TopicName");
```

### Arguments

*AccessName*

The existing Access Name to assign the new AppName and Topic Name values to. Actual string or message tag.

*NodeName*

The new Node Name to assign. Actual string or message tag.

*AppName*

The new Application Name to assign. Actual string or message tag.

*TopicName*

The new Topic Name to assign. Actual string or message tag.

### Remarks

The values assigned to the AccessName, AppName, and TopicName arguments can be specified as literal strings or string values provided by other InTouch tags or functions.

If you configured a secondary source for access name failover using the **Add Access Name** dialog box, you cannot change the secondary source configuration at run time using the IOSetAccessName() function.

**Note:** When creating Access Names in WindowMaker, if the Access Name is SuiteLink type, the InTouch HMI prevents Access Names from accessing the same, node, application, and topic. Do not allow the **IOSetAccessName()** function to redirect Access Names to duplicates during run time. Using the **IOSetAccessName()** function in run time allows SuiteLink type Access Names to be redirected to duplicate topics. The redirected Access Name will not work.

**Examples**

The **MyAccess1** Access Name can be changed to point to the **Excel** application and the **[Book1]Sheet1** topic, without affecting the current **NodeName**, by using the following script function:

```
IOSetAccessName("MyAccess1", "", "EXCEL", "[Book1]Sheet1");
```

If an empty string is specified for a Topic, the Access Name’s current Application value is updated and its Topic value is retained.

For example, the following script changes the Application Name for the **MyAccess2** Access Name to EXCEL without affecting its current Topic value:

```
IOSetAccessName("MyAccess2", "", "EXCEL", "");
```

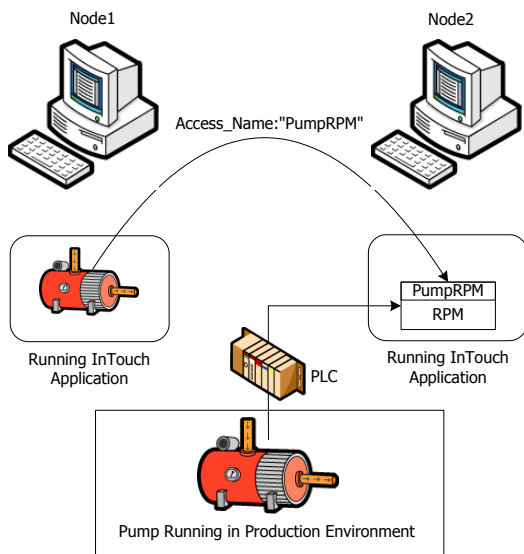
If an empty string is specified only for an Application Name, the tag’s current Topic value is updated and its Application value is retained. For example, the following script changes the Topic for the **MyAccess3** tag to **[Book2]Sheet1** without affecting its current Application Name value:

```
IOSetAccessName("MyAccess3", "", "", "[Book2]Sheet1");
```

This example can be used when PLC redundancy is a requirement.

## Converting Tags to Remote References

You can create distributed InTouch applications based upon a client-server architecture. Client applications can run on one network node that use tags defined on other remote nodes. The following figure shows an InTouch application running on Node 1 making a remote reference to the PumpRPM tag on Node 2.



In this example, you can retrieve the value of the PumpRPM tag from Node 2 in two ways:

- Create an I/O type tag in Node1's Tagname Dictionary that uses Node2 as the node name in the Access Name associated with the I/O tag.
- Use a direct remote reference to the **PumpRPM** tag. For example, **PLC1:"PumpRPM"**.

In a window or QuickScript, you can reference a remote tag by appending the Access Name as the prefix to the remote tag name in the form:

```
access_name:"tag_name"
```

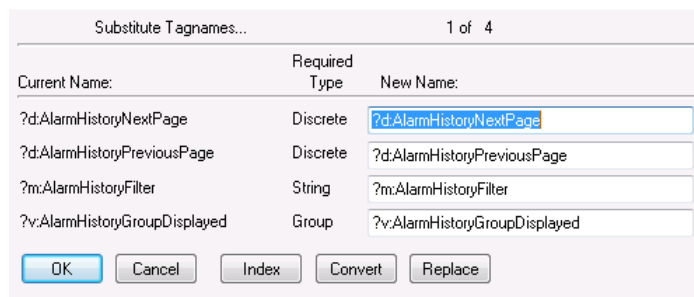
When importing a window or QuickScript, you can convert the placeholder tags to remote tag references. For example, you can convert the placeholder tags to point to the application from which you imported the window. The tags do not need to be defined in your local Tagname Dictionary.

You can use several methods to convert local tags to remote tag references:

- Append the remote tag reference
- Convert the placeholder tags associated with an imported window
- Launch the Tag Browser and open the tag source's Tagname Dictionary to select the remote tag reference.

### To manually convert tags to remote tag references

1. Open an application window in WindowMaker.
2. Select the object associated with the local tag that you want to change to a remote tag reference.
3. On the **Special** menu, click **Substitute Tags**. The **Substitute Tagnames** dialog box appears with a list of tags associated with the object.



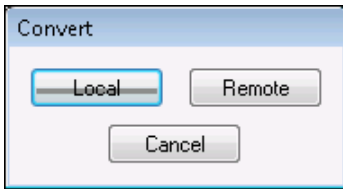
4. Click **Index** to add an index to each tag name.
5. Click **Convert**. The **Convert** dialog box appears with options to convert the tags to local or remote reference tags.
6. Click **Remote**. The **Access Names** dialog box appears. All Access Names defined in your local InTouch application are shown.
7. Select an Access Name from the list.
8. Click **Close**. All tags listed in the **Substitute Tags** dialog box are automatically converted to remote tag references with the Access Name appended to the tag name.
9. Click **OK**.

### To convert an imported window's tags to remote references

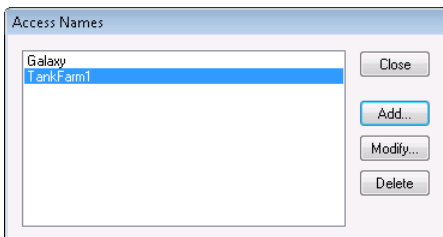
1. Open the imported window and select all objects.



2. On the **Special** menu, click **Substitute Tags**. The **Substitute Tagnames** dialog box appears.
3. Click **Convert**. The **Convert** dialog box appears.



4. Click **Remote**. The **Access Names** dialog box appears. All Access Names defined in your local InTouch application are shown.

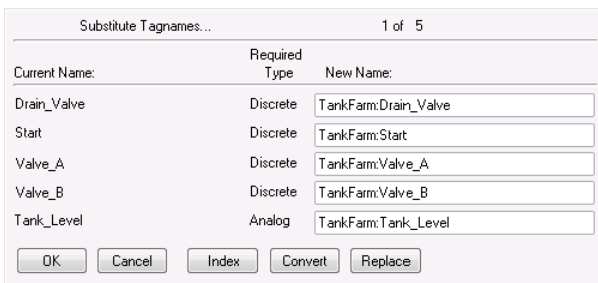


5. Select an Access Name from the list.

To verify the Access Name is correctly configured, click **Modify**.

If you do not have an Access Name currently defined that points to the tag source, click **Add** and define it. The Access Name must include the name of the remote node where the application resides.

6. Click **Close**. All tags listed in the **Substitute Tags** dialog box are automatically converted to remote tag references with the Access Name appended to the tag name.

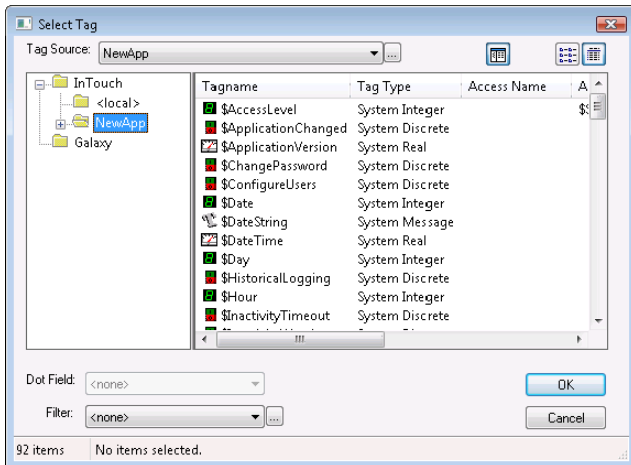


7. Click **OK**.

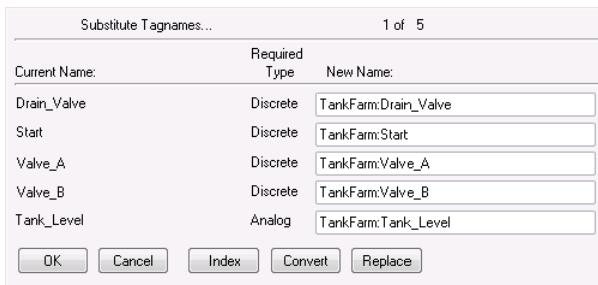
### To select a remote tag reference in the Tag Browser

1. Select the objects associated with the local tag that you want to convert to a remote tag reference.
2. On the **Special** menu, click **Substitute Tags**. The **Substitute Tagnames** dialog box appears showing the selected tags.
3. Delete the tag name in the **New Name** box that you want to replace with a remote tag reference.
4. Double-click the **New Name** box. The **Select Tag** dialog box appears with a list of tags associated with the application.
5. Select a remote tag using the Tree view.

- a. Click the **Tree** icon to show a hierarchical list of all local and remote Access Names in the left pane.



- b. Select a remote Access Name folder to show its assigned tags in the right pane.
- c. Select a remote tag that you want to use as a remote reference.
- d. Click **OK**. The **Substitute Tagnames** dialog box appears with the selected remote tag name in the **New Name** box.

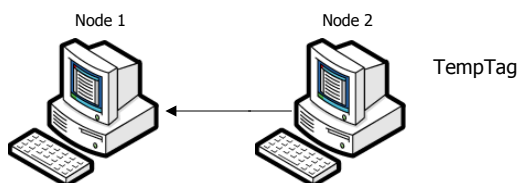


- 6. Click **OK** to close the dialog box and associate the remote tag with the selected objects.
- 7. Repeat these steps for each tag that you want to associate with a remote reference.

## Accessing I/O Data by Remote References

The InTouch HMI supports true client-server architecture for factory automation applications. You can design client applications without using any tags from the local Tagname Dictionary located on the same node as the running InTouch application. You can run an application on one node that uses tags from a remote node by using remote tag referencing.

The following figure shows a simple example where the **TempTag** is defined locally on Node2:



In this example, the InTouch application running on Node1 can retrieve the value of TempTag on Node2 by two methods:

- Create an I/O type tag in Node1's Tagname Dictionary that uses Node2 as the Node Name in the Access Name associated with the I/O tag.
- Use a remote reference directly to **TempTag**. For example, Node2:"TempTag".

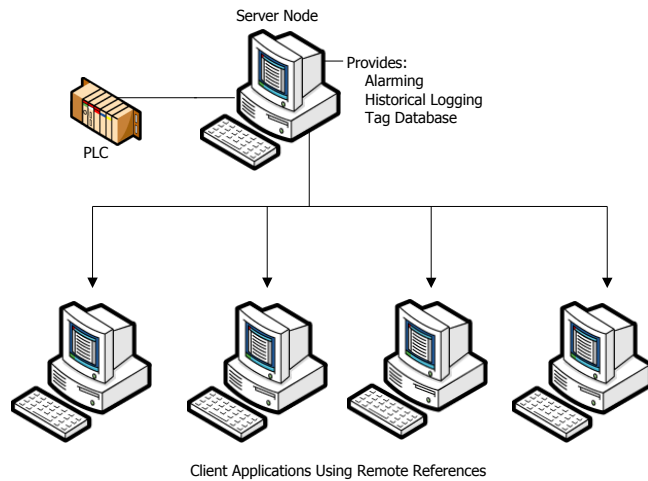
When you want to directly reference a remote tag in any other application, only AccessName:item is required. You do not have to define the remote tag in your local Tagname Dictionary. Remote references can also access data from any I/O data source such as a DAServer or Microsoft Excel.

You can also make a remote reference to SuperTags. The valid syntax for a remote tag reference to a SuperTag is:

*Access\_name:Parent\_Instance\ChildMember\SubMember.*

For more information about remote references to SuperTags, see *Referencing SuperTag Members* on page 126.

When importing a window or QuickScript, you can convert the placeholder tags to remote tag references. You do not have to define tags in your local Tagname Dictionary. The remote references are accessible from any application on the network.



## Redirecting Remote References During Run Time

You can redirect ArchestrA object references or remote references to InTouch tags at run time with a script. You can switch object instances for a graphic symbol based on certain conditions being met or directly by an operator action.

### IOSetRemoteReferences() Function

You can use the **IOSetRemoteReferences()** script function to redirect ArchestrA object references or remote references to tags while an InTouch application is running. **IOSetRemoteReferences()** finds all remote references that match specified strings and changes those references according to specified argument values. You can create a script that triggers the function to redirect references based on conditions being met or by a user action.

#### Category

Misc

## Syntax

```
IOSetRemoteReferences(BaseAccess, NewAccess, MatchString, SubstituteString, Mode)
```

## Arguments

### *BaseAccess*

This string argument specifies the original configured Access Name to match in the references.

### *NewAccess*

The new Access Name. The new Access Name is applied to all references in which the original Access Name matches the string provided with *BaseAccess* and for which the original Item Name matches the *MatchString* value if one is specified.

### *MatchString*

The string to match in the original configured Item Name in the references. If the *MatchString* value is an empty string, it is regarded as a match for any Item Name.

### *SubstituteString*

The string to substitute for the original Item Name. The string replaces the *MatchString* value to create the new active Item Name for the references. If *SubstituteString* is an empty string, no substitution is made.

### *Mode*

Determines the method used to compare the original configured Item Name to the *MatchString* value. Matching is always from the beginning of the Item Name. A *Mode* value of 0 specifies the match must be for the entire Item Name or up to a period (.) within the name. A *Mode* value of 1 specifies that a partial match is allowed, even if the next character is not a period.

## Remarks

**IOSetRemoteReferences()** does not check the validity of the new tag or Access Name before changing the object references.

- **IOSetRemoteReferences()** only changes remote references. The function redirects those references in which the original, configured Access Name matches the specified value of the *BaseAccess* argument, and the original Item Name matches the *MatchString* value.
- A single call to **IOSetRemoteReferences()** affects all remote references in active windows that are in memory in which the original, configured name strings match the values assigned to the *BaseAccess* and *MatchString* arguments.
- If you do not assign a value to the *BaseAccess* argument, **IOSetRemoteReferences()** does not redirect any remote references.
- If the *MatchString* argument is empty, **IOSetRemoteReferences()** redirects all remote references in which the original Access Name matches the value assigned to the *BaseAccess* argument.
- When the *Mode* argument is set to 0, replacement in the Item Name is only done for full object names (or tags), or full property names (or dotfields). The value of the *MatchString* argument must match the entire original Item Name or up to a character followed by a period.
- When the *Mode* argument is set to 1, partial replacement of the item string is allowed when the item string starts with the match item string. That is, *MatchString* must match some portion of the original item string, but that sub-portion must start at the beginning of the item string. The last character in the matching string does not need to be followed by a period.

- The original, configured names for a remote reference remain unchanged. Subsequent calls to **IOSetRemoteReferences()** do not need to recognize the current active name. Calls to **IOSetRemoteReferences()** can be made in any order.
- If you want two or more windows to refer to one remote reference, that remote reference acts like an I/O tag. When you redirect it, all windows see the same thing. Do not use a single name to refer to two separate targets at the same time.

---

**Note:** Changing many references simultaneously, for example in a Window OnShow script, can take some time before all references are resolved.

---

### Examples

The following example redirects all remote references to the pump001 item name of the Galaxy Access Name if the original item name exactly matches pumpX.

```
IOSetRemoteReferences("Galaxy", "", "pumpX", "pump001", 0);
```

The following example matches changes the Galaxy Access Name to TagServer1 if the item name exactly matches pumpX. Also, the item name is changed to p2.

```
IOSetRemoteReferences("Galaxy", "TagServer1", "pumpX", "p2", 0);
```

The following example changes the TagServer1 Access Name to TagServer2 when the item name is pumpX. Also, the item name is changed to backpump3.

```
IOSetRemoteReferences("TagServer1", "TagServer2", "pumpX", "backpump3", 0)
```

The following example changes the Tank item name of the TagServer1 Access Name to Plant.

```
IOSetRemoteReferences("TagServer1", "", "Tank", "Plant", 1)
```

The following example does not redirect any remote references because no value is assigned to the *BaseAccess* argument.

```
IOSetRemoteReferences("", "Galaxy", "pumpX", "pump001", 0);
```

### Restoring References

If the *NewAccess* argument is empty without an assigned value, **IOSetRemoteReferences()** restores the active Access Name to the original base Access Name.

If the *MatchString* argument is empty without an assigned value, **IOSetRemoteReferences()** restores the active Item Name to the original Item Name.

---

**Note:** Even if *SubstituteString* is not empty, if *MatchString* is empty, the Item Name is restored to the original Item name. Inserting text at the beginning of the name is not allowed. For example, running the script `IOSetRemoteReferences("Access1", "", "", "Valve", 0);` does not append the string Valve at the beginning or end of the all original Item Names.

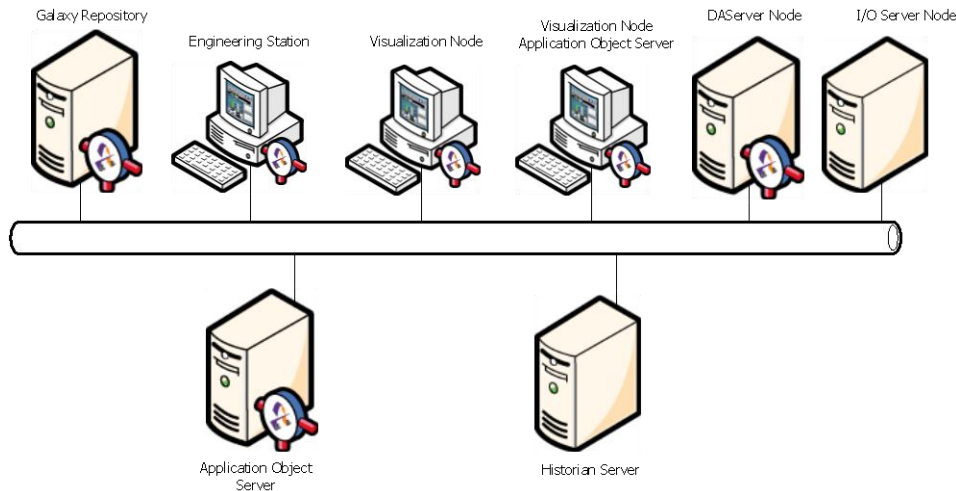
---

If *SubstituteString* is empty without an assigned value, **IOSetRemoteReferences()** restores the active Item Name to the original Item Name. Using a non-empty *MatchString* with an empty *SubstituteString* enables you to select a subset of remote references on the indicated original base access and restore them to their original Item names.

## Accessing Application Server Data from InTouch

ArchestrA provides a set of common services and underlying architecture for a suite of products. You can select from this array of products to build plant automation and information systems using modular ArchestrA components.

Application Server provides a set of services to build plant automation applications. Application Server services are distributed across a set of nodes within the system.



Typically, the InTouch HMI works with Application Server to provide the visual interface for the application that operators interact with to manage a plant process.

## Using Application Server Object Attributes with InTouch Tags

You can use InTouch tags to interact with Application Server object attributes to transfer data between an InTouch application and an Application Server data repository.

The InTouch HMI communications protocol support includes Message Exchange. When WindowViewer runs an InTouch application, Message Exchange regards WindowViewer as an anonymous engine.

This anonymity means the InTouch application has no attributes that can be accessed by other Message Exchange clients. WindowViewer is not configured, managed, or viewed as an AutomationObject within an Application Server Galaxy. The InTouch HMI only uses Message Exchange to subscribe to those active items available from Application Server.

You can use the InTouch Tag Browser to select a Galaxy as a tag source for remote tags and browse through the namespace of the Galaxy. An Application Server object attribute or property of an attribute can be used in a remote reference or used as an item for an InTouch I/O tag.

For more information about using Application Server objects as a remote tag source, see *Configuring the InTouch HMI to Use a Galaxy as a Remote Tag Source* on page 85.

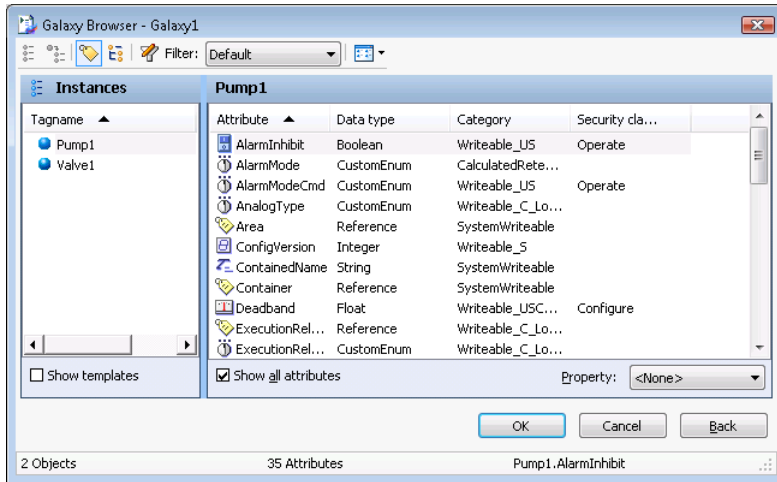
A pre-configured Access Name called Galaxy is available in WindowMaker for Message Exchange access. The Galaxy Access Name is only relevant for the InTouch HMI in the ArchestrA environment. There are no user configurable properties for the Galaxy Access Name.

## Browsing Application Server Object Attributes from InTouch

To browse and select Application Server attributes from the InTouch HMI, you must first set up a tag source for the Galaxy in the InTouch HMI. For more information, see *Configuring the InTouch HMI to Use a Galaxy as a Remote Tag Source* on page 85.

You select the tag source from within the InTouch **Select Tag** dialog box.

The InTouch Galaxy Browser dialog box lists all objects within the target Galaxy. You can expand an object to see its contained objects or run-time accessible attributes. The Galaxy Browser dialog box does not show those attributes that start with an "\_" (marked as hidden) or any attribute of type QualifiedStruct.



To return from the Galaxy Browser dialog box to the regular InTouch Select Tag dialog box, click **Back**.

## Application Server Browser Restrictions

The following restrictions apply when viewing Application Server objects with the InTouch Attribute Browser dialog box:

- Only run-time visible attributes in a single Galaxy’s namespace are viewable. This includes the capability to switch between an object’s TagName and its HierarchicalName.

An object attribute can be selected from the **Attribute Browser** dialog box if it meets the following requirements:

- Visible during run time
- From a currently checked in AutomationObject
- Attribute name does not have an "\_" following a "."
- The Attribute Browser dialog box only shows Application Server object attribute data types supported by the InTouch HMI. For more information about supported data types, see *Mapping Application Server Data Types to InTouch Data Types* on page 81.
- The InTouch Attribute Browser dialog box does not show any attributes that would result in an InTouch Item Name greater than the 95 character maximum.

- Automation object array elements can be displayed or retrieved in the InTouch HMI by using "TagName.AttributeName[index]" as a reference. Use an index of -1 to show or retrieve all array element values.
- You can select a property of an object attribute with the Tag Browser. By default, the Value property is selected when the attribute is selected.

## Special Extensions in Application Server Objects

The WindowMaker Tag Browser and the Message Exchange client in WindowViewer add and recognize special extensions to each Application Server object attribute. These extensions provide access to information that otherwise would not be available to the InTouch HMI.

These special extensions are optional and do not need to be used by the InTouch application. However, applications that handle status and quality information frequently need to use these extensions.

These items extend the namespace of attributes to include additional properties that WindowViewer can expose to application scripts and Windows. For example, the reference to "TIC101.PV.#ReadSts" provides access to the MxStatus information for the subscription to TIC101.PV. This information is very useful for displaying extended information that is exposed by Message Exchange.

These properties do not exist as object attributes in Application Server as named elements. The properties are client-side extensions, supplied in the client abstraction layer, that make object attributes visible to the InTouch HMI. The following table describes the attribute extensions for the InTouch HMI:

---

Attribute Extension	Data Type	Purpose
None	Coerced	The default extension. Means that no extension is provided. The item is read/written with the value data type coerced as appropriate to InTouch. Failed read/write information can be obtained if the client subscribes to the #ReadSts or #WriteSts items described below. Example: "Pump1.PV".

---



Attribute Extension	Data Type	Purpose
.#VString for floats / doubles attributes only:	String (read/write)	Sets up a subscription to the reference that has ".#VString" as a suffix. This is the underlying reference. Returns the current value of the underlying reference as a string when reads and writes are both working correctly. If the UserGetAttribute returns bad status, this item returns an abbreviated status description string based on MxStatus instead of the value. The abbreviated status description strings are:  "?Pending" - pending "?Warning" - warning "?Comms" - communication error "?Config" - configuration error "?Oper" - operational error "?Security" - security error "?Software" - software error "?Other" - other error

For .#VString, if the status is good but the quality is bad, this item returns the quality description string, available from Message Exchange, instead of the value.

The value is returned as a string only when quality and status for UserGetAttribute are both good or when the quality is good and the status is uncertain. This may require coercion if the data type returned by Message Exchange is not a string. When quality or status is uncertain, the value shows a "?" as a suffix. For example, "3.27?" or "True?".

## Mapping Application Server Data Types to InTouch Data Types

Application Server includes some attributes and data types that do not map directly to the four primary data types supported by InTouch tags.

The following table shows how the client abstraction layer maps data types for read and write operations. It also shows the data types that the Galaxy dictionary exposes to InTouch.

Attribute		
Property Data Type	InTouch Data Type	Notes
Float	Real - 32 bit	Pass through.

Attribute		
Property Data Type	InTouch Data Type	Notes
Double	Real - 32 bit	If double is IEEE NAN, then convert to float IEEE NAN. If this overflows, set <b>Quality</b> to Bad and pass float IEEE NAN. If the double fractional value is a smaller fraction than the smallest float fraction of 1.17549E-38, treat it as 0.0 float and set <b>Quality</b> to Good.
Boolean	Discrete	False = 0, True = 1.
Integer	Integer - 32 bit	Pass through.
String (always Unicode)	Message - MBCS (multi-byte character set encoded)	Truncate the string if it is too long for InTouch and set the quality to uncertain. Retain both bytes of each Unicode character.
Time	Message - MBCS	Format as an appropriate string for the locale. Use <b>MxValue</b> to convert the string.
ElapsedTime	Real	Pass as float seconds. <b>MxValue</b> supports coercion to this type.
MxDataType	Message - MBCS	Pass the string.
MxSecurityClassification	Message - MBCS	Pass the string.
MxQuality	Message - MBCS	Pass the string.
MxReference	Message - MBCS	Pass the reference string only as Unicode.
MxCategorizedStatus	Message - MBCS	Pass the string.
MxQualifiedStruct	Not supported	Not supported.
MxQualifiedEnum	Message - MBCS	Pass the Enum string. The integer ordinal value can be accessed by applications by referencing #EnumOrdinal. For example, "Pump1.PV.#EnumOrdinal".

Attribute		
Property Data Type	InTouch Data Type	Notes
Array of Strings	Message - MBCS (Read-only)	Put each element of the array into a comma-separated string such as:  "String1,String2,String3"  up to the maximum limit of an InTouch string value. If this is truncated, the associated quality sent to the InTouch HMI is uncertain. You cannot write to an entire array of strings, but you can write to individual elements of an array.
All arrays	Integer, Real, Message, or Discrete	Only supports a subscription to a single element of an array. In this case, the conversions described above are applicable. Otherwise, the return is an empty string with Bad quality.
MxInternationalizedText	Message	This is accessed as a string type at run time.

## Read/Write Behavior of Application Server Attributes

When the system writes to an Automation object attribute, its write status is initially set to "?Pending".

When the write is completed, the #WriteSts string is updated with the result of the write. If the write completes successfully, #WriteSts value is set to an empty string. If the write returns an error and is pending, the #WriteSts item continues to show the most recent write status even if subscription updates continue to occur on reads.

You can also use the #VString1 to #VString4 items to convert float values or double values to a string format, The number N indicates the number of decimal places to be returned. For example, "3.1234" is the string for #VString4. You can use the #VString item without a number to round the float or double value to an integer and to return it as a string value.

Attribute Extension	Data Type	Purpose
.#EnumOrdinal	Integer (read/write)	Contains the currently read ordinal value for attributes of the Qualified Enum type. This is a way to return an integer for enumerations rather than returning a string.

Attribute Extension	Data Type	Purpose
.#ReadSts	String (Read-only)	<p>Contains the current read status of the subscribed to item to which the string is concatenated. It appears as "TIC101.PV.#ReadSts". This is provided by Message Exchange as a string. Its' value can be one of the following:</p> <p>"?Config" - configuration error</p> <p>"?Comms" - communication error</p> <p>"?Oper" - operational error</p> <p>"?Pending" - pending</p> <p>"?Warning" - warning</p> <p>"?Security" - security error</p> <p>"?Software" - software error</p> <p>"?Other" - other error</p> <hr/> <p><b>Note:</b> If the associated item (for example, TIC101.PV) is not subscribed to, the returned string is blank.</p>
.#WriteSts	String (Read-Only)	<p>Contains the last write status of the item to which this string is concatenated, for example Pump1.Cmd.#WriteSts. This is provided by Message Exchange as a string. If the string is blank, the last write to the item is successful. Otherwise, #WriteSts can be one of the following values:</p> <p>"?Config" - configuration error</p> <p>"?Comms" - communication error</p> <p>"?Oper" - operational error</p> <p>"?Pending" - pending</p> <p>"?Warning" - warning</p> <p>"?Security" - security error</p> <p>"?Software" - software error</p> <p>"?Other" - other error</p> <hr/> <p><b>Note:</b> If the associated item (for example, TIC101.PV) is not subscribed to, the returned string is blank.</p>

## Configuring the InTouch HMI to Use a Galaxy as a Remote Tag Source

You can use the InTouch Tag Browser to select an Application Server object as a tag source and browse the Galaxy database. Application Server object attributes or attribute properties can be used in remote references or as items for InTouch I/O tags.

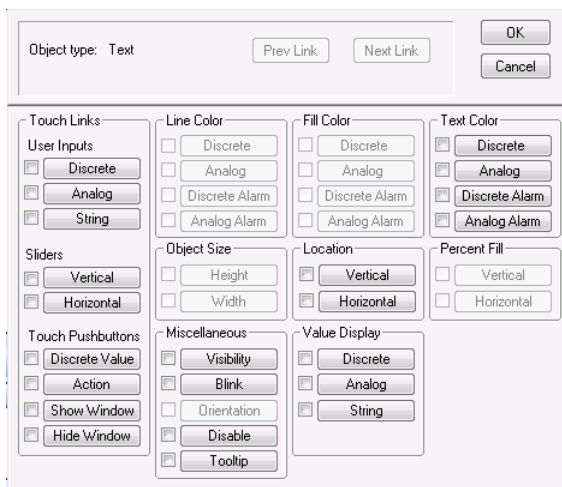
When an InTouch application runs as a client providing the visual interface for an Application Server application, you must install Application Server Bootstrap and a WinPlatform object on the same node as the InTouch application.

To browse the Galaxy namespace from the InTouch HMI, you also need to install the ArchestrA Integrated Development Environment (IDE).

The InTouch HMI uses the Message Exchange functionality of the Platform to view the Galaxy namespace and provide better data communication.

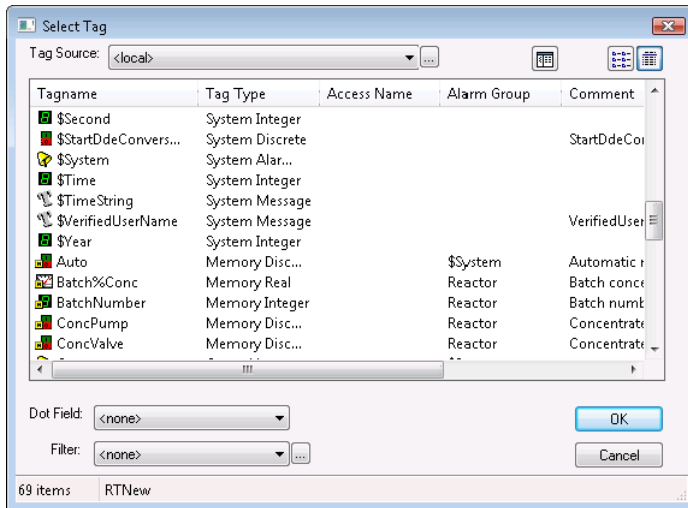
### To configure InTouch to use a Galaxy as a remote tag source

1. Open an application window in WindowMaker.
2. Double-click on a text object. The **Animation Links** dialog box appears.

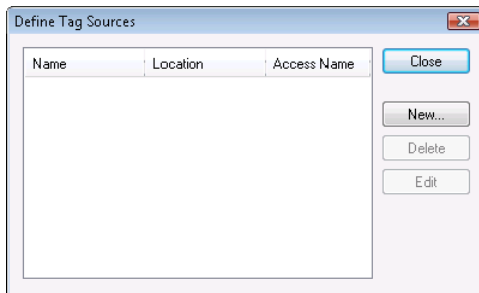


3. In the **Value Display** area, click **Analog**. A dialog box appears to insert an expression.
4. Delete any expression located within the **Expression** box.

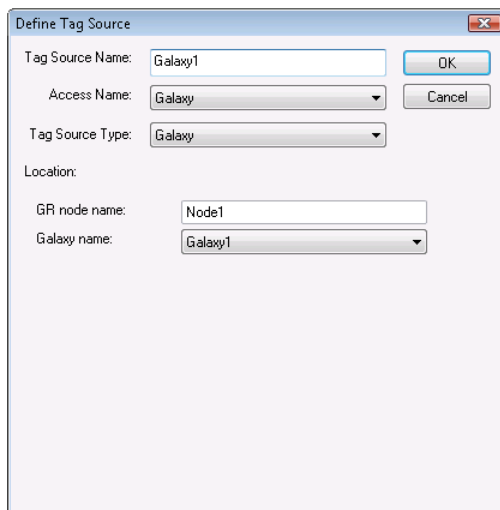
5. Double-click in the **Expression** box. The **Select Tag** dialog box appears.



6. Click the button to the right of the **Tag Source** box. The **Define Tag Sources** dialog box appears.

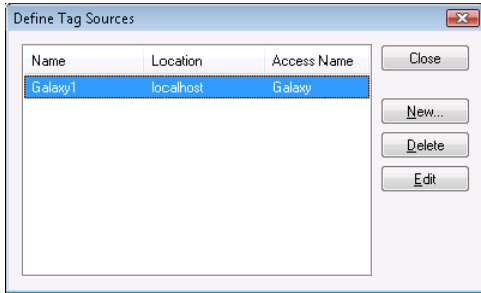


7. Click **New** to show the **Define Tag Source** dialog box.

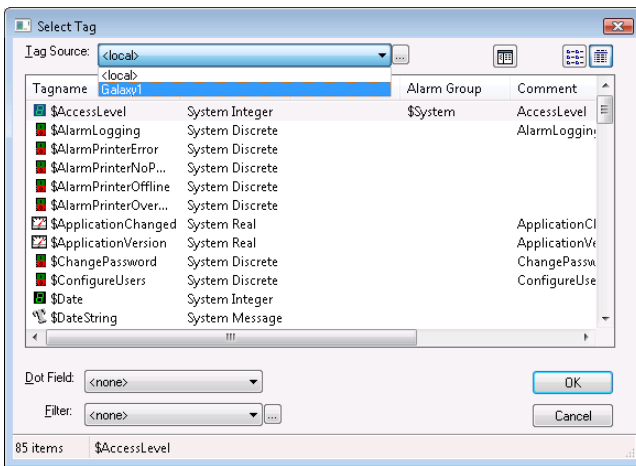


8. Enter values for the boxes of the **Define Tag Source** dialog box. Do the following:
  - a. In the **Tag Source Name** box, type the name of your remote Galaxy tag source.
  - b. In the **Access Name** box, select Galaxy from the list.
  - c. In the **Tag Source Type** box, select Galaxy from the list.

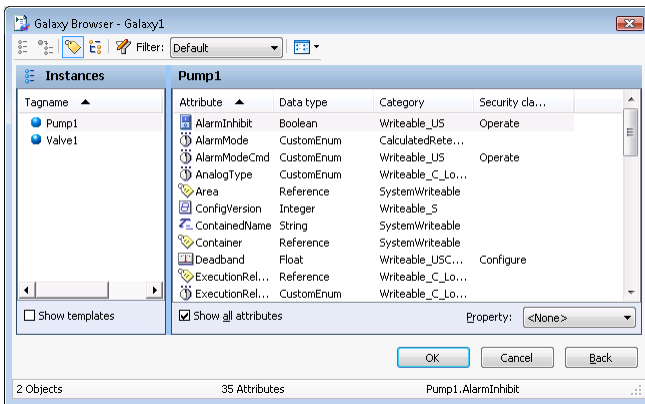
- d. In the **Location** area, type the Galaxy Repository Node name and select the Galaxy from the list.
- e. Click **OK**. The **Define Tag Sources** dialog box shows the remote tag source you defined in its list.



- 9. Click **Close** to close the **Define Tag Sources** dialog box. The **Select Tag** dialog box shows the new tag source from the list of the **Tag Source** box.

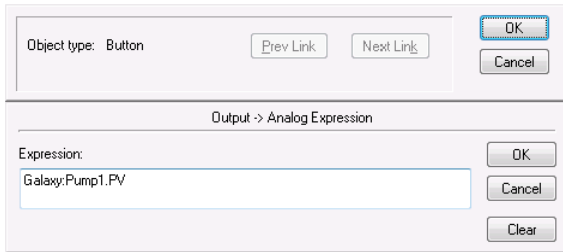


- 10. Select the new tag source you created from the **Tag Source** box. The **Galaxy Browser** dialog box opens with a list of tags in the left pane.



- 11. Select a tag from the left pane of the **Galaxy Browser** dialog box. The right pane of the **Galaxy Browser** dialog box refreshes with the attributes of the selected tag.

- Click the attribute you want to use and click OK. The **Output -> Analog Expression** dialog box appears with an expression in the **Expression** box.



- Confirm the string expression is correct.

The expression uses the form:

Galaxy:tag\_name.attribute\_name

**Example:**

Galaxy:Pump1.PV

- Click **OK** to close the **Output > Analog Expression** dialog box.
- Configure remaining object links as needed.
- Click **OK** in the animation link dialog box.
- Click **Runtime**. The text object shows the value for the configured tag attribute.

## Viewing Timestamp and Quality Information for an I/O Tag

The InTouch HMI places value, time, and quality (VTQ) indicators on all data values delivered to VTQ-aware clients. InTouch uses a set of dotfields as indicators of data quality that are useful for troubleshooting purposes.

- The **.Value** dotfield contains the value of the specified tag. This is also the default dotfield of every InTouch tag. If no other dotfield is specified, the **.Value** dotfield is assumed.
- The set of **Time** dotfields are time stamps indicating the last time a tag was updated.
- The **Quality** dotfields show the reliability of data values assigned to an I/O tag.

### Viewing Timestamp Information for an I/O Tag

The set of **Time** dotfields are assigned to tags in the following format:

Tag\_name.Time\_Dotfield

#### .TimeDate Dotfield

The **.TimeDate** dotfield shows the whole number of days that have passed between January 1, 1970 and the last update of a tag value from an I/O Server.

**Category**

Tag



**Usage**`Tag_name.TimeDate`**Parameter***Tag\_name*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Integer (read-only).

**See Also**`.TimeDateString`, `.TimeDay`, `.TimeDateTime`, `.TimeHour`, `.TimeMinute`, `.TimeMsec`, `.TimeMonth`, `.TimeSecond`, `.TimeTime`, `.TimeTimeString`, `.TimeYear`

## .TimeDateString Dotfield

The **.TimeDateString** dotfield shows the date and time when a tag value is updated from an I/O Server.**Category**

Tag

**Usage**`Tag.TimeDateString`**Parameter***Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Message (read-only).

**See Also**`.TimeDate`, `.TimeDay`, `.TimeDateTime`, `.TimeHour`, `.TimeMinute`, `.TimeMsec`, `.TimeMonth`, `.TimeSecond`, `.TimeTime`, `.TimeTimeString`, `.TimeYear`

## .TimeDateTime Dotfield

The **.TimeDateTime** dotfield shows the fractional number of days that have passed between January 1, 1970 and the last update of a tag value from an I/O Server.**Category**

Tag

**Usage**`Tag.TimeDateTime`**Parameter***Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Real (read-only).

**See Also**

.TimeDate, .TimeDateString, .TimeDay, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeDay Dotfield

The **.TimeDay** dotfield shows the number of days within the month that have passed since the last update of a tag value from an I/O Server.

**Category**

Tag

**Usage**

*Tag*.TimeDay

**Parameter**

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Integer (read-only).

**Valid Values**

Values can range from 1-31.

**See Also**

.TimeDate, .TimeDateString, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeHour Dotfield

The **.TimeHour** dotfield shows the number of hours within a day that have passed since the last update of a tag value from an I/O Server.

**Category**

Tag

**Usage**

*Tag*.TimeHour

**Parameter**

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Integer (read-only).

### Valid Values

Values can range from 0-23.

### See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeMinute Dotfield

The **.TimeMinute** dotfield shows the minute portion of the time when the tag value was last updated from an I/O Server.

### Category

Tag

### Usage

```
Tag.TimeMinute
```

### Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

### Data Type

Integer (read-only).

### Valid Values

Values can range from 0-59.

### See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeMonth Dotfield

The **.TimeMonth** dotfield shows the month number (1-12) of the date when a tag value is updated from an I/O Server.

### Category

Tag

### Usage

```
Tag.TimeMonth
```

### Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

### Data Type

Integer (read-only).

## Valid Values

Values can range from 1-12.

## See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeMsec Dotfield

The **.TimeMsec** dotfield shows the millisecond portion of the time when the tag value was last updated from an I/O Server.

## Category

Tag

## Usage

*Tag*.TimeMsec

## Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

## Data Type

Integer (read-only).

## Valid Values

Values can range from 0 - 999.

## See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString, .TimeYear

## .TimeSecond Dotfield

The **.TimeSecond** dotfield shows the second portion of the time when the tag value was last updated from an I/O Server.

## Category

Tag

## Usage

*Tag*.TimeSecond

## Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

## Data Type

Integer (read-only).

## Valid Values

Values can range from 0 - 59.

## See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeTime, .TimeTimeString, .TimeYear

## .TimeTime Dotfield

The **.TimeTime** dotfield shows the timestamp when a tag value is updated from an I/O Server expressed as the number of milliseconds that have elapsed since midnight.

## Category

Tag

## Usage

```
Tag.TimeTime
```

## Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

## Data Type

Integer (read-only).

## Valid Values

Values can range from 0 - 86399999.

## See Also

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTimeString, .TimeYear

## .TimeTimeString Dotfield

The **.TimeTimeString** dotfield shows the time when a tag value is updated from an I/O Server.

## Category

Tag

## Usage

```
Tag.TimeTimeString
```

## Parameter

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

## Data Type

Message (read-only).

**See Also**

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeYear

**.TimeYear Dotfield**

The **.TimeYear** dotfield shows the four-digit year when a tag value is updated from an I/O Server.

**Category**

Tag

**Usage**

*Tag*.TimeYear

**Parameter**

*Tag*

Any discrete, integer, real, message, indirect analog, indirect discrete, or indirect message tag.

**Data Type**

Integer (read-only).

**Valid Values**

Any year expressed as a four-digit number.

**See Also**

.TimeDate, .TimeDateString, .TimeDay, .TimeDateTime, .TimeHour, .TimeMinute, .TimeMsec, .TimeMonth, .TimeSecond, .TimeTime, .TimeTimeString.

**Viewing Quality Information for an I/O Tag**

You can use a set of quality dotfields to ensure the integrity of data sent between an I/O Server and your InTouch applications. Quality dotfields represent the quality state of an item's data value. This quality attribute makes it fairly easy to monitor the integrity of InTouch data sent between network nodes.

The data quality standard is based on the OLE for Process Control (OPC) proposed standard, which in turn is based on Fieldbus Data Quality Specifications.

You can configure how you want numeric values formatted at run time based on data type and data quality.

**Quality Data Format**

An I/O Server can report six mutually exclusive states of data quality sent to clients by assigning values to a set of InTouch .Quality dotfields. The low eight bits (Least Significant Byte) of the Quality dotfields are currently defined in the form of three bit fields; Quality (Q), Substatus (S), and Limit (L) with the following format:

**QQSSSLL**. When the client application cannot communicate with the server, the **.Quality** dotfield's value is 0.

The data quality states reported by an I/O Server with .Quality dotfields are shown in the following table:

Quality States	Decimal Value	Hex Value	MS Byte XXXXXXXX	LS Byte QQSSSSL	Quality	Quality Sub Status	Limit
Good	192	0x00C0	00000000	11000000	Q=3	S=0	L=0
Clamped High (Out of Range)	86	0x0056	00000000	01010110	Q=1	S=5	L=2
Clamped Low (Out of Range)	85	0x0055	00000000	01010101	Q=1	S=5	L=1
Cannot Convert	64	0x0040	00000000	01000000	Q=1	S=0	L=0
Communications Failed	24	0x0018	00000000	00011000	Q=0	S=6	L=0
Cannot Access Point	4	0x0004	00000000	00000100	Q=0	S=1	L=0

## About Data Quality Dotfields

The .Quality dotfields indicate the quality of data values the last time data was received. The SuiteLink and DDE protocols only send clients (for example, WindowViewer) updated quality when a data change is provided by the I/O Servers. Therefore, you only observe a quality change when a new data value is received. Some I/O Servers can send current data values with updated quality when the quality associated with the data changes.

It might not be possible to directly reference the quality of a server item's value using the SuiteLink and DDE protocols. To do this, the I/O Server must directly support Item.Quality. Without this support, the item fails to go on advisement and the .Quality dotfield value never changes from 0.

The TestProt I/O Server simulator does not directly support Item.Quality. The simulator does not send out new data values when you modify the quality using the Quality menu command.

If you want to observe data quality for an I/O item and the I/O Server does not directly support addressing of Item.Quality, then define an InTouch I/O tag to look at the server item and then monitor the .Quality of the InTouch tag. If you exceed your tag limit, then consider using the IOSetRemoteReferences() function in a script to dynamically adjust the I/O points.

The SuiteLink and DDE protocols do not interpret connection state or other changes in I/O Server status as quality items sent to the client. As a result, an item's quality does not necessarily indicate the current data server status nor the current status of the client-to-server connection. An I/O Server process can stop and the value of the .Quality field does not change. If the communications link is lost, the value of the .Quality field does not necessarily change.

Use DDE or SuiteLink internal status items to monitor the I/O Server connection.

## .Quality Dotfield

The .Quality dotfield shows a numerical assessment of the quality of data provided by an I/O Server.

## Category

Tag

## Usage

*Tag*.Quality

## Parameter

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

## Data Type

Integer (read-only).

## Valid Values

Values can range from 0-255.

## See Also

.QualityLimit, .QualityStatus, .QualitySubstatus

## Example

```
IF I0Tag.Quality <> 192 THEN
    LogMessage("This data is not Good, assuming high-byte of 2-byte quality is zero");
    LogMessage("Better to check .QualityStatus to avoid this assumption");
ENDIF;
```

## .QualityLimit Dotfield

The **.QualityLimit** dotfield shows the quality limit of a data value provided by a connected I/O Server.

## Category

Tag

## Usage

*Tag*.QualityLimit

## Parameter

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

## Data Type

Integer (read-only).

## Valid Values

0 = Not Limited

1 = Low Limited

2 = High Limited

3 = Constant



**See Also**

.Quality

**.QualityLimitString Dotfield**

The **.QualityLimitString** dotfield shows the quality limit string of a data value provided by a connected I/O Server.

**Category**

Tag

**Usage**

*Tag*.QualityLimitString

**Parameter**

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

**Data Type**

Message (read-only).

**See Also**

.Quality, .QualityLimit

**.QualityStatus Dotfield**

The **.QualityStatus** dotfield shows an integer quality status of a data value provided by an I/O Server.

**Category**

Tag

**Usage**

*Tag*.QualityStatus

**Parameter**

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

**Data Type**

Integer (read-only).

**Valid Values (SSSS)**

0 = Bad

1 = Uncertain

3 = Good

**Example**

```
IF I0Tag.QualityStatus <> 3 THEN
    LogMessage("This data is not Good!");
```

```
ENDIF;
```

### See Also

.Quality, .QualitySubStatus

## .QualityStatusString Dotfield

The **.QualityStatusString** dotfield shows the quality status string of a data value provided by an I/O Server.

### Category

Tag

### Usage

```
Tag.QualityStatusString
```

### Parameter

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

### Data Type

Message (read-only).

### See Also

.QualityStatus, .QualitySubStatus, .Quality

## .QualitySubstatus Dotfield

The **.QualitySubstatus** dotfield shows the quality sub-status of a data value provided by an I/O Server.

### Category

Tag

### Usage

```
Tag.QualitySubstatus
```

### Parameter

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

### Data Type

Integer (read-only).

### Valid Values (SSSS) and (QQ)

Substatus (SSSS) for BAD quality (QQ=0).

0 = Non-specific

1 = Configuration error

2 = Not connected

3 = Device failure

- 4 = Sensor failure
- 5 = Last known value
- 6 = Communication failure
- 7 = Out of service

Substatus (SSSS) for UNCERTAIN quality (QQ=1).

- 0 = Non-specific
- 1 = Last usable value
- 4 = Sensor not accurate
- 5 = Engineering Units exceeded
- 6 = Sub-Normal

Substatus (SSSS) for GOOD quality (QQ=3).

- 0 = Non-specific
- 6 = Local override

**See Also**

.QualityStatus, .QualitySubStatus, .Quality

## .QualitySubstatusString Dotfield

The **.QualitySubstatusString** dotfield shows the quality sub-status string of a data value provided by an I/O Server.

**Category**

Tag

**Usage**

*Tag*.QualitySubstatusString

**Parameter**

*Tag*

Any discrete, integer, real, indirect analog, or message tag type.

**Data Type**

Message (read-only).

**See Also**

.QualityStatus, .QualitySubStatus, .Quality

## Initializing and Resetting I/O Connections at Run Time

WindowViewer initiates all I/O conversations when it starts the InTouch application. You can also manually restart I/O conversations while your InTouch application is running. You can initialize and reset I/O connections during run time with WindowViewer commands or scripts.

You can also specify that I/O connections should be reinitialized based on their default values. If you select this option, the default settings are used and the current settings are ignored when Access Names are reinitialized. To reinitialize I/O conversations by Access Name, you must have an InTouch application with Access Names already defined.

## Reinitializing I/O Connections with Commands

The **Special** menu in WindowViewer includes a set of commands to reinitialize all I/O conversations or select a specific I/O conversation.

You can reinitialize Access Names using the default settings of InTouch. Using default reinitialization, the Access Name ignores the current values assigned to node name, application name, and topic. The Access Name is reinitialized with the original Access Name settings.

### To reinitialize all Access Names at run time

1. On the **Special** menu, click **Reinitialize I/O**.
2. Click **Reinitialize All**. All Access Names are reinitialized.

### To reinitialize selected Access Names at run time

1. On the **Special** menu, click **Reinitialize I/O**, then click **Select**. The **Reinitialize I/O** dialog box shows a list of Access Names.

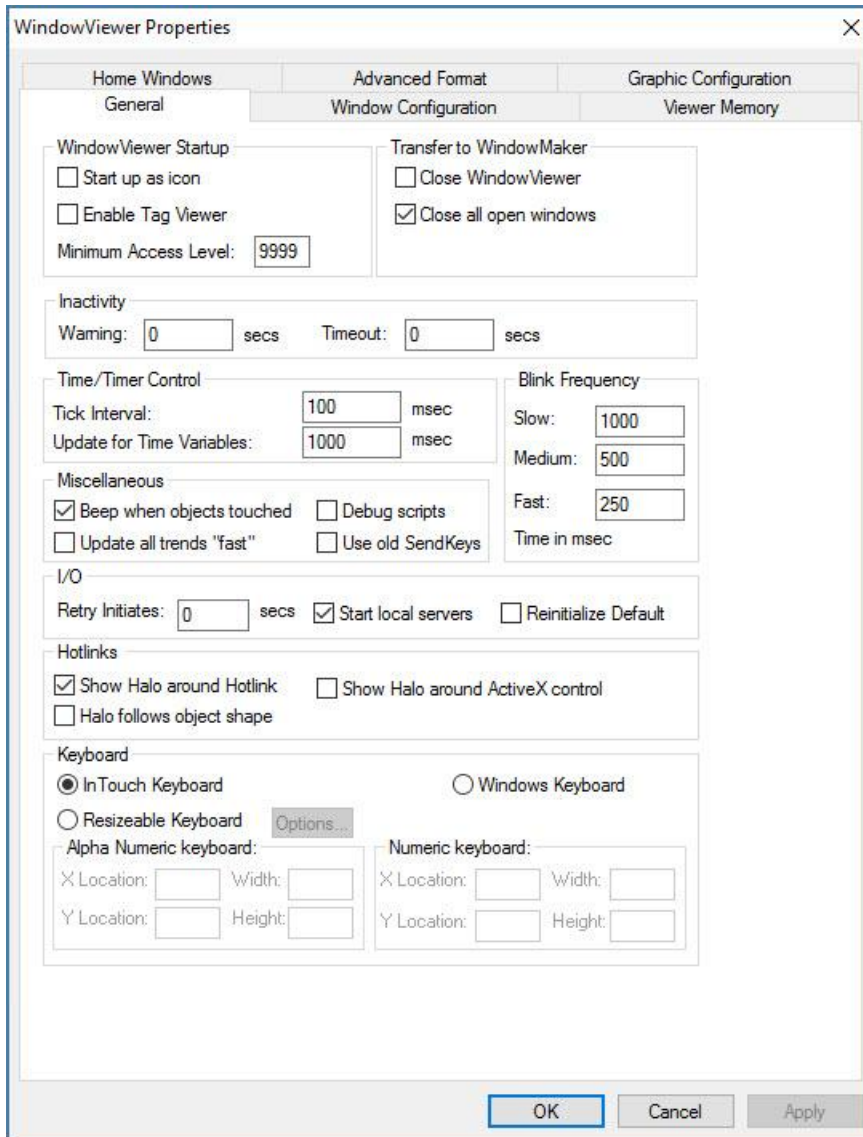


2. Click one or more Access Names to reinitialize, then click **Reinitialize**. The selected Access Names are reinitialized.

### To reinitialize Access Names using default settings

1. Open an application within WindowMaker.

- On the **Special** menu, click **Configure**, then click WindowViewer. The WindowViewer **Properties** dialog box appears with a list of options.



- On the **General** tab, select the **Reinitialize Default** check box in the **I/O** area.
- Click **OK**.
- Open the application in WindowViewer.

- On the **Special** menu, click Reinitialize I/O, then click Select. The **Reinitialize I/O** dialog box appears.



- Select one or more Access Names to reinitialize, then click Reinitialize. The current settings are ignored for node name, application name, and topic. The Access Names are reinitialized with the original Access Name settings.

## Reinitialize I/O Connections with Scripts

You can reinitialize an I/O connection to one or more Access Names by creating a script that includes the following functions:

- **IOReinitAccessName()**
- **IOReinitialize()**
- **IOStartUninitConversations()**

### IOReinitAccessName() Function

The **IOReinitAccessName()** function reinitializes the I/O connection to a specified Access Name.

#### Category

I/O communication

#### Syntax

```
IOReinitAccessName("AccessName", Default);
```

#### Arguments

*AccessName*

Access Name to be reinitialized.

*Default*

Default = 1. The I/O reinitialization uses the original default Access Name values assigned from WindowMaker.

Default = 0. The I/O reinitialization uses the current node, application, and topic values assigned to the Access Name.

#### Remarks

The default settings are determined by the settings in the Access Name configuration panel and also in the WindowViewer configuration (Retry Initiates, Start Local Servers, Reinitialize Default).

## Examples

This example reinitializes the I/O connection to `AccessName1` using the default values assigned to the node, application, and topic.

```
IOReinitAccessName("AccessName1", 1);
```

This example reinitializes the I/O connection to `AccessName2` using the current values assigned to the node, application, and topic.

```
IOReinitAccessName("AccessName2", 0);
```

## IOReinitialize() Function

The **IOReinitialize()** function first closes and then restarts all active I/O connections defined for an InTouch application.

### Category

Miscellaneous

### Syntax

```
IOReinitialize();
```

### Arguments

None.

### Remarks

The **IOReinitialize()** function performs the same operation as the **Reinitialize I/O** command on the WindowViewer **Special** menu.

If WindowViewer is running as a service, the **IOReinitialize()** function and **Reinitialize ALL** command on the WindowViewer **Special** menu do not reinitialize all Access Names. If you select the Access Names listed in the **Reinitialize I/O** dialog box and click **Reinitialize**, the selected Access Names are reinitialized.

For more information about navigating to the **Reinitialize I/O** dialog box, see *Reinitializing I/O Connections with Commands* on page 100.

### Example

This example closes any active I/O connections and restarts all I/O connections defined for the InTouch application.

```
IOReinitialize();
```

## IOStartUninitConversations() Function

When WindowViewer begins running an InTouch application, it automatically processes an initiate request to start all I/O conversations. If an I/O Server program does not respond to WindowViewer's initiate request, you can use the **IOStartUninitConversations()** script function to force WindowViewer to attempt to start the I/O conversation again.

### Category

Miscellaneous

### Syntax

```
IOStartUninitConversations();
```

### Arguments

None.

### Remarks

The **IOStartUninitConversations()** function performs the same operation as the **Start Uninitiated Conversations** command on the WindowViewer **Special** menu.

### Example

This example forces WindowViewer to submit another initiate request to start all I/O connections defined for the InTouch application.

```
IOStartUninitConversations();
```

## Using Failover Functionality with Access Names

You can specify that the InTouch HMI automatically switches to a secondary I/O Server if the primary I/O Server experiences communication problems. This is called I/O failover.

### Configuring Failover

You can specify that your InTouch application switches to a failover secondary I/O Server when it can no longer communicate with the primary I/O Server.

When you set the failover, you specify the failover deadband. The failover deadband is the delay in seconds before switching from the primary Access Name to the secondary Access Name. The InTouch HMI triggers the failover when the expression or an I/O communication failure is true for the length of the deadband period. When the failover deadband is set to 0 or blank, the failover is triggered as soon as an I/O communication failure is detected.

#### To configure failover for an Access Name

1. If needed, stop WindowViewer.
2. On the **Special** menu, click **Access Names**. The **Access Names** dialog box appears with a list of all defined Access Names.
3. Select the Access Name from the list to add a failover server.
4. Click **Modify**. The **Modify Access Names** dialog box appears.



- Click **Enable Secondary Source**. The **Modify Access Name** dialog box expands.

- Do the following:
  - In the **Node Name** box, type the node name of the secondary I/O Server.
  - In the **Application Name** box, type the program name of the secondary I/O Server program from which data will be acquired.
  - In the **Topic Name** box, type the topic name you want to access from the secondary I/O source.
  - In the **Which protocol to use** area, select either **DDE** or **SuiteLink** as the secondary I/O Server communication protocol.
  - In the **When to advise server** area, select **Advise all items** or **Advise only active items** for the secondary I/O source.
- Click **Failover**. The **Failover Configuration** dialog box appears.

- Enter an optional failover expression or double-click in the **Failover expression** box to select a tag. For more information about failover expressions, see *Forcing Failover to a Backup Access Name* on page 106.
- In the **Failover Deadband** box, type the length of the failover deadband in seconds.
- Select **Switch back to primary when Failover conditions clear** if you want to enable switching from the secondary Access Name back to the primary Access Name after the failover condition clears.

The default is to not switch back to the primary Access Name. If you select **Switch back to primary when failover conditions clear**, then the **Fail-back Deadband** option becomes selectable from the **Failover Configuration** dialog box.

11. From **Failback Deadband**, type the length of the failback deadband in seconds.

The InTouch HMI triggers a failback to the primary Access Name after the expression and any associated I/O communication failure clear for the deadband period. When the expression is left blank or 0, the fail-back occurs as soon as the I/O communication failure condition clears.

12. Click **OK** to close the **Failover Configuration** dialog box.
13. Click **OK** to close the **Modify Access Name** dialog box.

## Editing the Access Name Parameters of a Failover Pair

To edit Access Name parameters that are part of a failover pair, you must have an Access Name configured for failover with a secondary I/O source.

### To edit the Access Name parameters of a failover pair

1. If needed, stop WindowViewer.
2. On the **Special** menu, click **Access Names**. The **Access Names** dialog box appears.
3. Select the Access Name pair and click **Modify**. The **Modify Access Name** dialog box shows the parameters for the primary and secondary Access Names.
4. Change the Access Name parameters for the primary and secondary Access Names.
5. Click **OK** to close the **Modify Access Name** dialog box.
6. Click **Close** to close the **Access Names** dialog box.

## Removing Failover for an Access Name

To remove failover for an Access Name, you must have an Access Name configured for failover with a secondary I/O source.

### To remove failover for an Access Name pair

1. On the **Special** menu, click **Access Names**.
2. Select the Access Name pair and click **Modify**. The **Modify Access Name** dialog box appears.
3. Clear the **Enable Secondary Source** check box.
4. Click **OK**. Failover for the Access Name pair is disabled.

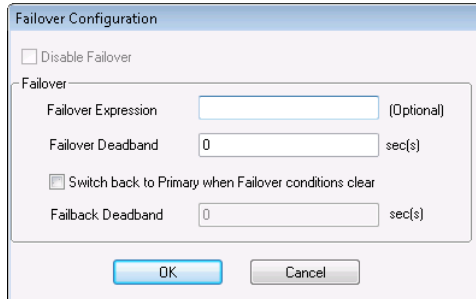
## Forcing Failover to a Backup Access Name

You can manually switch between the primary and secondary sources of an Access Name without experiencing a failover. This is called a forced failover. To force a failover, you must have an Access Name configured for failover with a secondary I/O source.

You can use a failover expression or the **IOForceFailover()** script function to force a failover.

## Failover Expression

The **Failover Configuration** dialog box shows the **Failover Expression** option to include a tag or expression that triggers a failover. The figure below shows the dialog box with the failover memory discrete tag entered as the value of **Failover Expression**.



Setting the failover expression to true, for example by setting the failover tag to true, switches the Access Name from the primary (false) to secondary (true) I/O data sources.

## IOForceFailover() Function

The **IOForceFailover()** script function switches between the primary and secondary data sources of the Access Name. The active I/O node toggles between the primary and secondary nodes with each invocation of the script function.

Typically, the **IOForceFailover()** function is part of a script associated with button or another window object. Operators select the object from an application window to force a failover. After operators click the object again, the **IOForceFailover** function forces the I/O connection back to the formerly active I/O node.

### Category

I/O Communication

### Syntax

```
IOForceFailover("AccessName");
```

### Argument

*AccessName*

Access name for which failover has been configured.

### Example

The Acc1 Access Name has Primary and Secondary data sources and Primary is active. Acc1 fails over to the Secondary data source when the script runs.

```
IOForceFailOver("Acc1");
```

## Temporarily Disabling Failover Functionality

You can manually disable failover switching between the primary and secondary I/O nodes of an Access Name. A typical case for temporarily disabling failover is during the brief period when the components of an InTouch system are started and not yet ready. After the components have stabilized, you can restore failover switching.

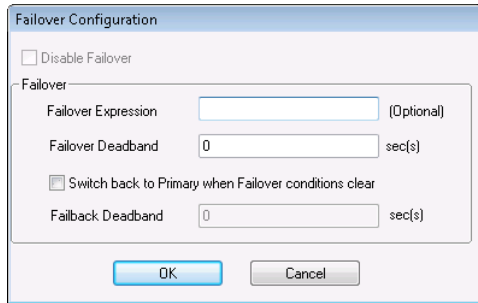
To disable failover for an Access Name, you must have an Access Name configured for failover with a secondary I/O source.

You can manually disable failover switching by two methods:

- Select the **Disable Failover** option from the **Failover Configuration** dialog box.
- Run a script that includes the **IODisableFailover()** function.

## Disable Failover Configuration Option

The **Failover Configuration** dialog box includes the **Disable Failover** option to prevent the Access Name from switching between the primary and secondary I/O nodes.



You must edit the Access Name definition to set the **Disable Failover** option to active. As long as the option is active for the Access Name, failover is disabled.

## IODisableFailover() Script Function

You can use the **IODisableFailover()** function in a script to disable failover for a specified Access Name. **IODisableFailover()** disables switching for all failover methods except by the **IOForceFailover()** script function method.

### Category

I/O Communication

### Syntax

```
IODisableFailover ("AccessName",Option);
```

### Arguments

*AccessName*

Access name for which failover has been configured.

*Option*

1 = Disables failover

0 = Enables failover

### Remarks

The Access Name can be specified as a literal string or it can be a string value provided by other InTouch tags or functions.

### Examples

In this example, failover is disabled for the ModbusPLC1 Access Name.

```
IODisableFailover ("ModbusPLC1",1)
```

In this example, failover is enabled for the ModbusPLC1 Access Name.

```
IODisableFailover ("ModbusPLC1",0)
```

## Retrieving Information About Failover Pairs Using Scripting

You can write scripts that include functions that return the status of the primary, secondary, and active I/O sources of an Access Name. Typically, operators run a script to determine the status of an Access Name's secondary I/O source before forcing a failover.

To create scripts to return Access Name information, you must have an Access Name configured for failover with a secondary I/O source.

### IOGetAccessNameStatus() Function

The **IOGetAccessNameStatus()** script function returns an integer indicating the connection status of the primary, secondary, or active I/O source of an Access Name.

Typically, the **IOGetAccessNameStatus()** return value is associated with an integer tag. The value of the tag can drive a discrete value display animation link that shows the status of the Access Names's active, primary, and secondary I/O sources to an operator.

#### Category

Miscellaneous

#### Syntax

```
Result=IOGetAccessNameStatus("AccessName", Mode);
```

#### Arguments

*AccessName*

The existing Access Name for which to return the status.

*Mode*

The value assigned to this argument determines what Access Name of the failover pair is queried about its current status.

0 - Status of the active Access Name I/O source

1 - Status of the Access Name primary I/O source

2 - Status of the Access Name secondary I/O source

#### Results

##### Returned

Value	Description
-1	There is a configuration error in the Access Name. Either the Access Name does not exist or the Access Name does not have a defined secondary I/O source.
0	The connection to the requested I/O source is not successful.
1	The connection to the requested I/O source is successful.

**Remarks**

The **IOGetAccessNameStatus()** function is typically used in a script that determines the status of the secondary IO Source that is currently inactive. The operator runs the script to verify the status of the secondary connection before forcing a fail-over.

**Example**

This example returns the status of the secondary I/O source of the ModbusPLC1 Access Name. The returned value is associated with the **ANStatus** tag.

```
ANStatus = IOGetAccessNameStatus ("ModbusPLC1",2)
```

**IOGetActiveSourceName() Function**

The **IOGetActiveSourceName()** script function returns whether an access name currently uses its primary or secondary data source.

Typically, the **IOGetActiveSourceName()** function is included in a script associated with button or another window object. Operators then select the object from an application window to request the status of the application’s I/O Servers.

**Category**

Miscellaneous

**Syntax**

```
Result=IOGetActiveSourceName("AccessName");
```

**Argument**

*AccessName*

The existing Access Name for which to return the source name.

**Remarks**

**IOGetActiveSourceName()** returns a string that indicates whether the primary or secondary nodes of an Access Name are being actively polled. Possible return values of the **IOGetActiveSourceName()** function are:

- Primary      The primary node of the Access Name is actively polled.
- Secondary    The secondary or failover node of the Access Name is actively polled.
- Null          Both the primary and secondary nodes of an Access Name are inactive.

**Example**

In this example, the **ActiveServer** message tag is assigned the returned value (Primary, Secondary, or Null) that identifies the current active node of the ModbusPLC1 Access Name.

```
ActiveServer = IOGetActiveSourceName ("ModbusPLC1");
```

## Monitoring the Status of an I/O Connection

WindowViewer includes a built-in topic called IOStatus to monitor the status of a specific I/O conversation between an InTouch application and an I/O Server communicating with a PLC.

---

**Note:** In versions of InTouch before 7.0, the topic name was DDEStatus.

---

You can set up the **IOStatus** topic to monitor I/O conversations.

### Using IOStatus Topic Name

You can prepare the **IOStatus** topic to monitor I/O communication between WindowViewer and an I/O Server. In this example, WindowViewer communicates with the Simulation I/O Server to a PLC defined in the I/O Server with PLC1 as its topic name.

---

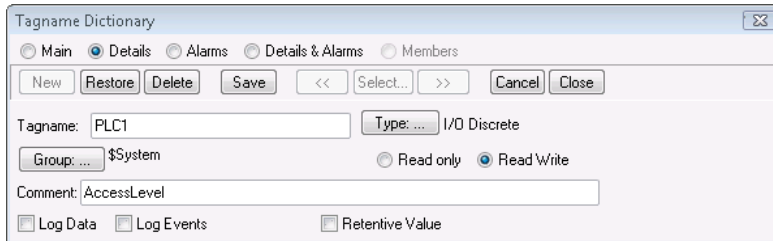
**Note:** The Simulation Server is a generic DAServer used as a training tool. The Simulation Server is located in the c:\program files\common files\Archestra folder.

---

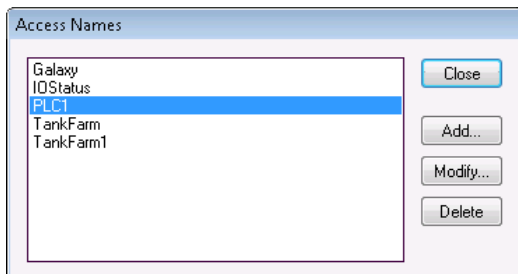
#### To monitor the status of I/O communications

1. Open an application in WindowMaker.
2. Open the Tagname Dictionary.
3. Create an I/O discrete tag.

When you are monitoring a I/O conversation using IOStatus, you must define at least one I/O type tag to the Access Name being monitored.



4. Click **Access Name** to assign the tag to an Access Name definition that defines IOStatus as its topic name.



Notice the **PLC1** Access Name definition currently exists.

5. Select PLC1 and click **Modify**.

Finding the Access Name containing the right topic name in this example is easy because the tag and Topic Name are the same.

6. Click **Cancel** to close the dialog box and return to the initial **Access Name** dialog box.
7. Click **Add**. The **Add Access Name** dialog box appears.

8. Do the following:
  - a. In the **Access box**, type IOStatus.
  - b. In the **Application Name** box, type **View** because you are going to monitor the status from **WindowViewer**.
  - c. In the **Topic Name** box, type **IOStatus** as the InTouch internal topic.
  - d. Select **Advise only active items**.
9. Click **OK** to close the dialog box. The initial **Access Name** dialog box reappears showing your new Access Name, IOStatus, in the list:



10. Click **Close** to close the dialog box and associate the new Access Name with your I/O Discrete tag.

In the Item box, type the Access Name for the actual topic name that you want to monitor.

11. Because your tag is the same as the Topic Name, you can select Use Tagname as Item Name and automatically enter it as the Item.

---

**Note:** When using the built-in topic IOStatus (DDEStatus before InTouch Version 7.0) to monitor an I/O conversation, the name you type in the Access Name box is always also used for the Item.

---

## Using IOStatus Topic Name in Excel

You can use Excel to monitor I/O status by entering the same information as a spreadsheet cell formula. For example, to monitor the same topic described in the previous procedure, enter the following formula in a cell:  
`=view|IOStatus!'PLC1'`

## Monitoring I/O Server Communications Status

For each topic name being used, you can use a built-in discrete item, **Status**, to monitor communication status with the I/O Server program. The **Status** item is set to 0 when a communication failure occurs. The **Status** item is set to 1 when communicating normally with the I/O Server program.

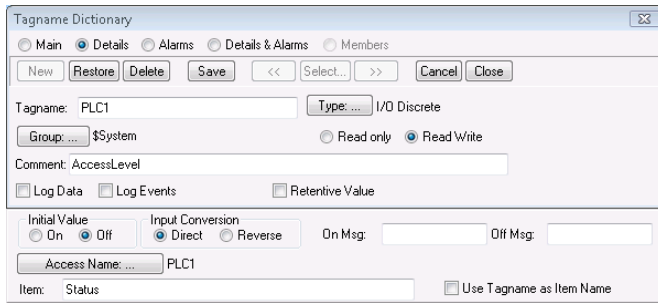
---

**Note:** When you monitor the status of a topic using the IOStatus item, at least one I/O point must be active for the monitored topic.

---

From the InTouch HMI, you can read the state of the server communications by defining a tag and associating it with the topic configured for the device by using the word **Status** as the Item Name. For example, if WindowViewer is communicating with a PLC using the Modbus DAServer, the Access Name definition is similar to the following example:

To monitor the status of all communication to the topic PLC1, create the following tag definition:



In Excel, you can read the status of the PLC communications by entering the following formula in a cell:  
`=SIMULATE | PLC1! 'STATUS'`

## Accessing InTouch Tag Data from Other Applications

When another application requests a data value from the InTouch HMI, it also must know three I/O address items. Follow these InTouch I/O address conventions.

VIEW (application name) identifies the InTouch run-time program that contains the data element.

TAGNAME (topic name) is the word always used when reading/writing to a tag.

ActualTagname (Item Name) is the actual tag defined for the item in the InTouch Tagname Dictionary.

For example, to access a data value in the InTouch HMI from Excel, a DDE Remote Reference formula is specified for the cell in which the data is written:

`=VIEW|TAGNAME!'ActualTag_name'`

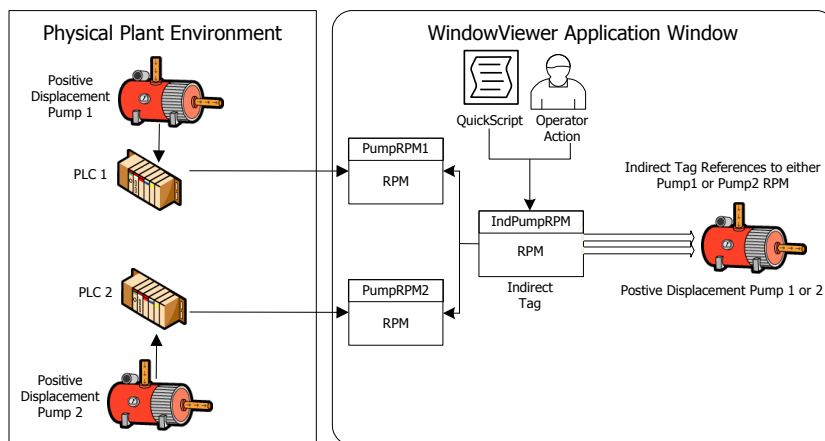
## Chapter 6

# Defining Indirect Tags

## About Defining Indirect Tags

Using indirect tags, you can create applications with window objects that show values from multiple tags.

The figure below shows a pump object within an application window. The pump object represents two possible process pumps based upon values set from an indirect tag. A QuickScript or operator action selects the source tag associated with the indirect tag.



Indirect tags minimize your development time. You create fewer application windows because a single window object can represent multiple processes running in the production environment.

## Using Indirect Tags with Scripts

You can use scripts to assign input source tags to an indirect tag. You assign an input source tag to an indirect tag by assigning the source tag's name to the indirect tag's **.Name** dotfield.

For example, if you create an indirect analog tag called **IndPumpRPM**, the two source **PumpRPM** tags are assigned to it with script statements similar to the following example:

```
IF PumpNo == 1 THEN
    IndPumpRPM.Name = "PumpRPM1";
ELSE
    IndPumpRPM.Name = "PumpRPM2";
ENDIF;
```

The indirect tag assignment script can be triggered by an application event or an operator action like clicking a window button.

When you equate an indirect tag to another source tag, the indirect tag behaves as if it is the source tag. If the value of the source tag changes, the indirect tag reflects the change. If the indirect tag's value changes, the source tag changes accordingly.

Because the .Name dotfield of an indirect tag is a simple string, you can dynamically assign the indirect tag target at run time. For example, if you create a Data Change QuickScript that runs each time the value of the **Number** tag changes, the source tag assigned to the indirect **IndPumpRPM** tag changes accordingly:

```
IndPumpRPM.Name = "PumpRPM" + Text(Number, "#" );
```

When this script runs, the value of the analog tag **Number** is converted to text and appended to the string **PumpRPM**. **If Number equals 1, this sets the name of the indirect IndPumpRPM tag to PumpRPM1.**

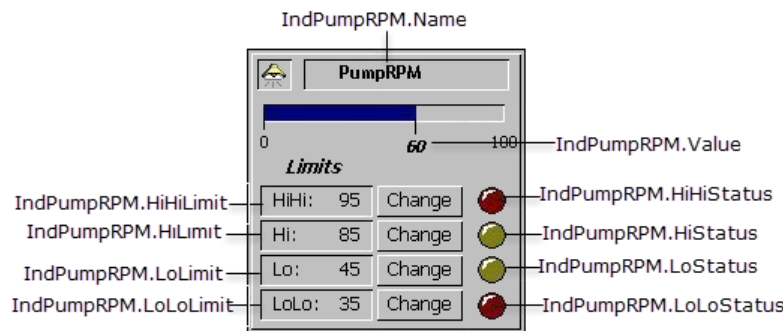
Indirect analog-type tags are used for both integer and real tags. Indirect tags can be mapped to any other tag as long they are the same tag type.

You can also assign retentive attributes to indirect tags. With retention, the indirect tag retains its most recent tag assignment when the application starts again.

## Using Indirect Tags with Local Tags

Indirect tags are typically used with tags defined in a local Tagname Dictionary. An indirect local tag enables you to create visual objects that show multiple attributes of a local tag. For example, you can create a faceplate within an application window. The faceplate contains selectable items with links to an indirect local tag assigned to different dotfields. In the following example, operators modify dotfield alarm limits linked to a local indirect **PumpRPM** tag.

The following figure shows a faceplate with animation links to pump RPM alarm limits. An indirect tag assigns the attributes of different local tags to the alarm limit faceplate.



To redirect the faceplate to the appropriate tag, include a statement within a QuickScript.

```
Indirect_tag_name.Name = "tag_name";
```

In this script example, tag\_name is the name of an actual tag defined in the local Tagname Dictionary. When the script runs, all dotfield values associated with this local tag become accessible to the application object through the indirect tag.

## Using Indirect Tags with Remote References

Remote indirect tag references differ from local tag references. The syntax for a remote reference is:

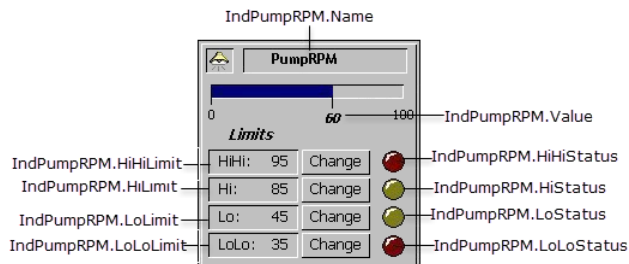
AccessName:Item

where

- *AccessName* is any valid InTouch Access Name.
- *Item* is any valid Item Name that is supported by the I/O Server specified in the Access Name definition.

When you use remote references, the server returns a value to the client, not a tag structure. The value includes a time stamp and a quality stamp. Thus, an indirect tag assigned to a remote reference cannot access any tag dotfields other than those related to value, time, and quality. For example, an indirect tag cannot access tag attributes through a remote reference to specify alarm limits.

One possible solution is to create a faceplate with a set of indirect tags. The following figure shows a faceplate to modify the alarm limits for a pump.



In this example, the faceplate uses 10 indirect tags that are associated with an implied **.Value** dotfield. The alarm faceplate is being redirected to the remote reference tag, IndPumpRPM, on a remote InTouch node named TagServer1. An InTouch Access Name is configured as follows:

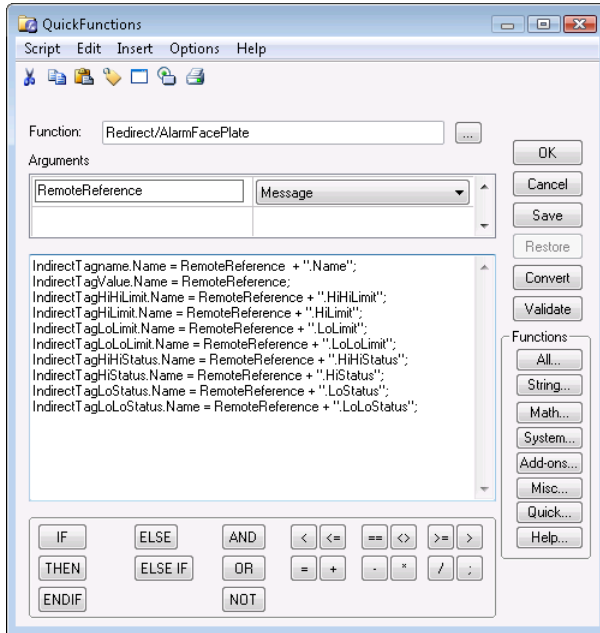
**Access Name:** TagSource1  
**Node Name:** TagServer1  
**Application Name:** View  
**Topic Name:** Tagname

To redirect the faceplate to the remote reference tag PumpRPM, run the following QuickScript:

```
IndPumpRPMName.Name = "TagSource1:PumpRPM.Name";
IndPumpRPMValue.Name = "TagSource1:PumpRPM";
IndPumpRPMHiHiLimit.Name = "TagSource1:PumpRPM.HiHiLimit";
IndPumpRPMHiLimit.Name = "TagSource1:PumpRPM.HiLimit";
IndPumpRPMLoLimit.Name = "TagSource1:PumpRPM.LoLimit";
IndPumpRPMLoLoLimit.Name = "TagSource1:PumpRPM.LoLoLimit";
IndPumpRPMHiHiStatus.Name = "TagSource1:PumpRPM.HiHiStatus";
IndPumpRPMHiStatus.Name = "TagSource1:PumpRPM.HiStatus";
IndPumpRPMLoStatus.Name = "TagSource1:PumpRPM.LoStatus";
IndPumpRPMLoLoStatus.Name = "TagSource1:PumpRPM.LoLoStatus";
```

The script must run each time the faceplate is redirected. Another solution is to create an InTouch QuickFunction that enables you to write a single script and pass it the name of the remote reference. You can reduce the amount of script coding by using multiple faceplates that call the same QuickFunction.

For example, using a similar set of script commands, you can define a QuickFunction called **RedirectAlarmFacePlate**:



You can call the **RedirectAlarmFacePlate** function to handle the entire redirection. To do this, the function must be called by another InTouch QuickScript. For example:

```
CALL RedirectAlarmFacePlate ("TagSource1:PumpRPM");
```

## Chapter 7

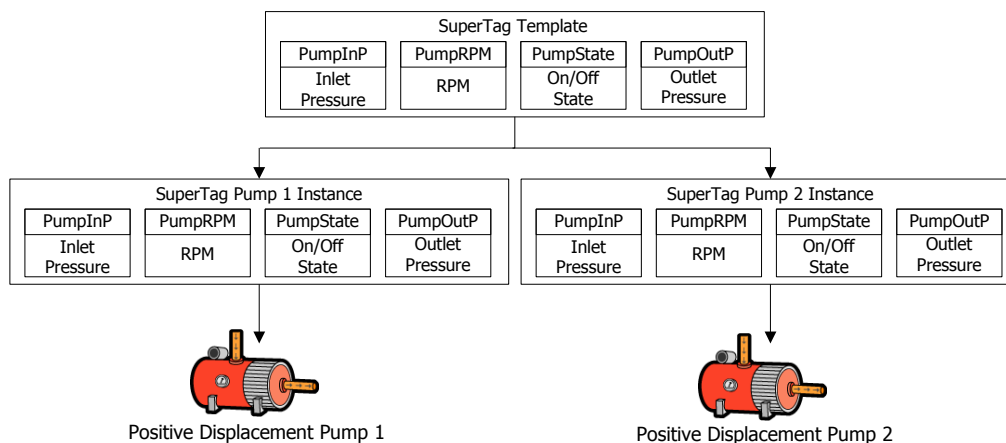
# Defining Reusable Tag Structures

## About Defining Reusable Tag Structures

A SuperTag is a template for a set of related tags. The tags that belong to a SuperTag template are associated with the common properties of a component in a manufacturing process.

SuperTags save development time. Instead of creating a set of tags for every component in the manufacturing process, you can replicate a single SuperTag template and create individual instances for all process components that have the same properties.

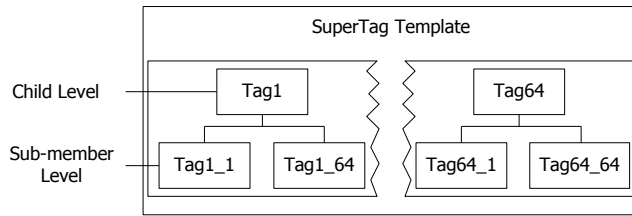
The figure below shows a single SuperTag template consisting of a set of related tags associated with pump data. The template can be replicated to create instances of the SuperTag for all of the identical pumps in the process.



All tags that belong to an instance of a SuperTag behave exactly like normal tags. The member tags can be assigned as InTouch discrete, integer, real, and message data types. The tags support trending, alarming, and all tag dotfields.

A SuperTag template can organize its member tags in two nesting levels. A SuperTag template can contain up to 64 embedded child tags. Each child tag can contain up to 64 sub-member tags. This gives you a total of 4095 tags in a SuperTag template.

The following figure shows how tags are organized within a SuperTag template.



When one SuperTag template parent is embedded into another SuperTag template, the embedded tag becomes a child member.

After you create a SuperTag parent template, the Tagname Dictionary lists it as a tag type in the **Tag Types** dialog box. The template can be selected immediately as a tag type when you create a new tag. You do not need to restart WindowMaker to define tags that use a newly created SuperTag type.

All SuperTag templates are saved in the supertag.dat file in the C:\Documents and Settings\All Users\Application Data\Wonderware\InTouch folder.

## Defining a SuperTag Template

You use the TemplateMaker utility to create, edit, and delete SuperTag templates and their member tags. After you create a SuperTag template the Tagname Dictionary **Tag Types** dialog box lists the SuperTag template and an indirect tag form of the template as tag types.

### To create a SuperTag template

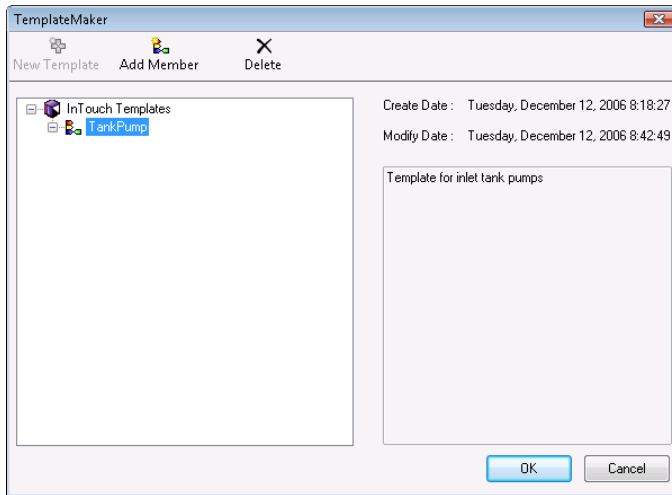
1. On the **Special** menu, click **TemplateMaker**. The **TemplateMaker** dialog box appears.
2. Click **New Template**. The **New Template** dialog box appears.
3. Enter a unique name for the SuperTag template in the **Name** box.

A template name can be up to 10 alphanumeric characters. The first character of a template name must be an alphabetical letter.

4. Optionally, type a comment that describes the template in the **Description** box.



- Click **OK**. The **TemplateMaker** dialog box appears again with the new template listed in the window. The **Add Member** and **Delete** buttons are active after a template is added.



### To add tags to a SuperTag template

- Open the **TemplateMaker** dialog box.
- Select the SuperTag template from the list.
- Click **Add Member** to show the **New Member Tag** dialog box.
- Do the following:
  - In the **Name** box, type the name of the member tag to be added to the template.

A member tag name can be up to 10 alphanumeric characters. The first character must be a letter.

---

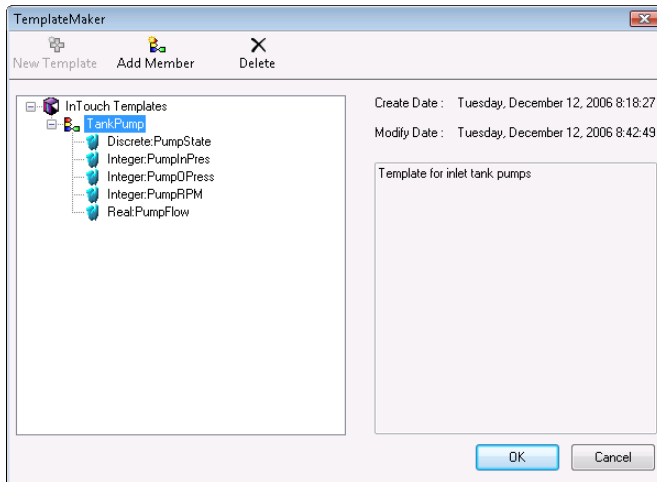
**Note:** Carefully consider the length of your tag names if you will use remote references. For more information about restrictions to SuperTag references, see *Referencing SuperTag Members* on page 126.

---

- In the **Type** box, type the tag type for the member or click **Type** and select the tag type from the list.
 

A tag type can be discrete, integer, real, message, or another SuperTag template. By default, all member tags are set to memory types when you define them in the TemplateMaker. However, when you define a template instance in the Tagname Dictionary, you must specify whether the template members remain as memory tags or be changed to I/O tags.
- Optionally, type a comment that describes the member tag in the **Comment** box.

- Click **OK**. The name of the new member tag appears beneath the SuperTag parent template in the **TemplateMaker** dialog box.



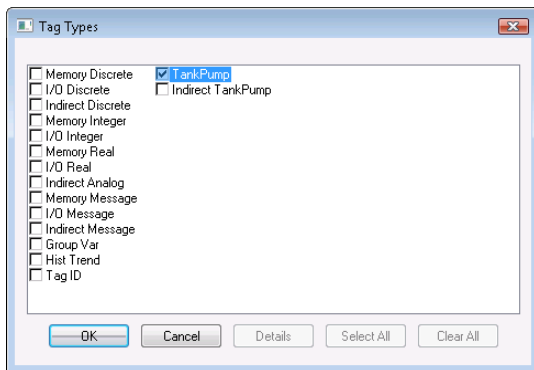
- The tag name includes the selected tag type assigned to the tag as a colon-delimited prefix.

**Example:**

Discrete:PumpState

- Repeat these steps to add all tags to the SuperTag template. The **TemplateMaker** dialog box lists all tags that you added beneath the SuperTag template name.

After creating a SuperTag template, the **Tag Types** dialog box shows both the SuperTag template and its indirect form as selectable tag types.



## Editing SuperTag Templates and Member Tags

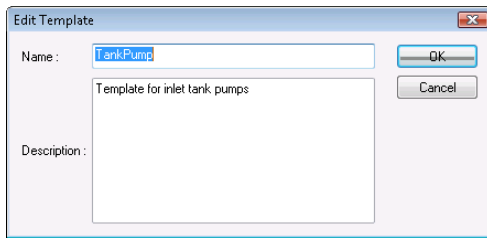
You can modify SuperTag templates or member tags at any time. Existing SuperTag instances derived from a template do not inherit the changes to the template. However, all new instances inherit the changes of the modified template.

You can delete an entire SuperTag template or selected member tags that belong to the template. Deleted templates are no longer listed in the Tagname Dictionary **Tag Types** dialog box.

**To edit an existing SuperTag template or member tag**

- Open the **TemplateMaker** dialog box.

2. Double-click the SuperTag template name or member tag that you want to edit. The **Edit Template** or **Edit Member Tag** dialog box appears. The dialog box shows either the SuperTag template's or member tag's current definition.



3. Make your changes.
4. Click **OK**.
5. The **TemplateMaker** dialog box refreshes and shows the date and time when editing changes were made.

#### To delete a SuperTag template or member tag

1. Open the **TemplateMaker** dialog box.
2. Select the SuperTag template name or member tag that you want to delete.
3. Click **Delete**. A message requests confirmation to delete the selected item.
4. Click **Yes** to delete the selected name.

The TemplateMaker list refreshes and removes the deleted item.

---

**Important:** Member tags of an instance of a SuperTag template cannot be deleted. For example, if PumpRPM is a member tag of the TankPump SuperTag template, it cannot be deleted from any instance of TankPump. You can only delete tags from the SuperTag template.

---

## Creating Instances of SuperTags

SuperTag templates and a template instance are different. An instance is an actual implementation of a SuperTag template in an InTouch application.

The most important difference between a template and an instance is that the parent template name is replaced by the name of the instance tag. The child template names and the sub-member tags do not change.

## Using the Tagname Dictionary to Create a SuperTag Instance

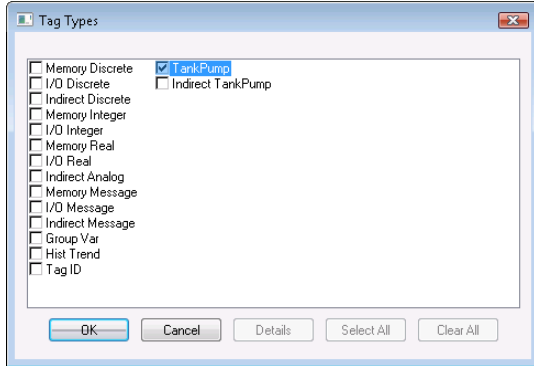
You can use the Tagname Dictionary to create a SuperTag instance. The Tagname Dictionary automatically creates all member tags and sub-member tags when you define the new SuperTag instance.

#### To create a SuperTag instance from a template

1. Open the Tagname Dictionary.
2. Click **New**.
3. In the **Tagname** box, type the name you want for the new SuperTag instance.

A SuperTag instance name can be up to 10 characters. An instance name follows the same naming rules as regular InTouch tags.

4. Click **Type** to show the **Tagname Types** dialog box.
5. Select a SuperTag template name from the list.

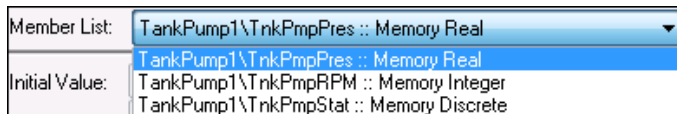


6. Click **OK**. The **Tagname Dictionary** dialog box expands to show additional options.



The new tag that you entered in the **Tagname** box becomes the parent for all member tags that belong to the selected SuperTag template.

7. Set the properties of the tag. Do the following:
  - a. In the **Member List** box, select a tag from the SuperTag template list.



- b. From **Data Access**, select **Memory** or **I/O** to show the respective Memory or I/O details dialog box.
  - c. Enter the details as you do for a standard InTouch tag.
  - d. Select the remaining member tags from the list and configure them.
8. Click **Close** after you specify all details for the member tags that belong to the SuperTag instance.

## Using the Tagname Dictionary to Replicate a SuperTag Instance

You can use the Tagname Dictionary to replicate an existing instance. After you replicate an instance, the tags become immediately available for use in animation links and InTouch QuickScripts.

### To replicate a SuperTag instance from the Tagname Dictionary

1. Open the Tagname Dictionary.
2. Click **Select** to show the **Select Tag** dialog box with a list of tags defined for the application.
3. Select a SuperTag instance from the list to use as the template for your new instance.
4. Click **OK**. The name of the selected template appears in the **Tagname** box.
5. Click **New**. A message requests confirmation to replicate the SuperTag instance.

6. Click **Yes**. The **Enter Name** dialog box appears and you are prompted for the name of the new SuperTag.
7. Enter a name up to 10 characters using standard tag naming conventions.
8. Click **OK**. The new SuperTag instance appears in the Tagname Dictionary.
9. If needed, edit the member tags as you do for a normal InTouch tag.
10. Click **Close**.

## Using the Tagname Dictionary to Add a Tag to a SuperTag Instance

You can add a tag as a member of an existing SuperTag instance using the Tagname Dictionary.

When you add a tag, you enter the exact name of your SuperTag instance followed by the backslash (\) delimiter and the name of the new member tag. For example:

Pump\_8\PumpSTS

---

**Note:** If you plan to use the ArcestrA Bulk Import Utility to migrate the tags from your InTouch application to Application Server, see *Importing SuperTags with the Bulk Import Utility* on page 127 for more information about substituting the standard backslash delimiter.

---

### To add a tag to a SuperTag instance

1. Open the Tagname Dictionary.
2. To add a tag to a SuperTag instance, do the following:
  - a. Click **New**.
  - b. In the **Tagname** box, type the exact name of your SuperTag instance followed by the backslash (\) delimiter and the name of the new member tag.
  - c. Click **Type**.
  - d. Select the tag type for the new member tag you are adding to the instance.
  - e. Click **OK**. The details dialog box for the member tag's type appears.
  - f. Enter the required details as you do for a normal InTouch tag.
3. Click **Save**.
4. Click **Select**.
5. Select the SuperTag instance that you added a member tag.
6. Click **OK**.

In the **Member List** box, all member tags that belong to the SuperTag template are listed.

Member List:	TankPump1\TnkPmpPres :: Memory Real
Initial Value:	TankPump1\TnkPmpRPM :: Memory Integer
Eng Units:	TankPump1\TnkPmpStat :: Memory Discrete
	TankPump1\TnkPmpVol :: Memory Real

The new member tag appears in the list.

## Other Ways to Create SuperTags

In addition to the TemplateMaker, you can also create SuperTags using the following methods:

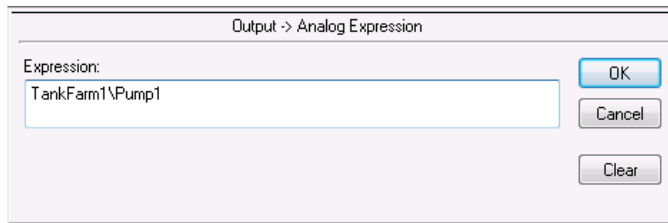
- In the Animation link expression input box
- Within InTouch QuickScripts.
- In an external file that you load into an application with the InTouch DBLoad utility

---

**Note:** When you use an alternative method to create the member, TemplateMaker does not show the member in the SuperTag template definition.

---

When you create a SuperTag through an animation expression or InTouch QuickScript, you must use the valid SuperTag format. For example:




---

**Note:** If the SuperTag instance and member tag you specify in an animation expression or QuickScript are not currently defined, you are prompted to define the tag. Click OK. The Tagname Dictionary appears and shows the SuperTag instance and member tag that you created.

---

The following syntax examples are valid:

ParentInstance\ChildMember ParentInstance\ChildMember\Submember

The following syntax examples are not valid:

ParentInstance\  
ParentInstance\ChildMember\

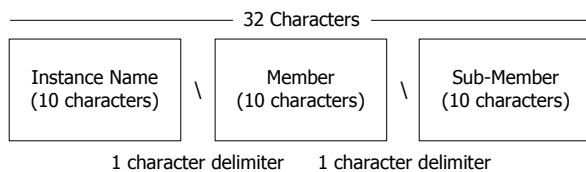
If an invalid format is used, an error message box indicates the SuperTag syntax contains an error.

## Referencing SuperTag Members

InTouch tag names can be a maximum of 32 characters. Each SuperTag

ParentInstance\ChildMember\Sub-member can be a maximum of 32 characters. The maximum length of a tag name places a restriction on references to SuperTags.

A SuperTag reference can be a maximum of two templates (ParentInstance\ChildMember) and one member deep, as shown in the following figure.



Each member in a SuperTag template is accessible in the standard format currently used to access the dotfields of standard InTouch tag types. The SuperTag reference syntax is supported throughout InTouch where standard tags can be used. For example, a valid SuperTag dotfield reference is:

```
TankFarm\Tank1\Pump1RPM.RawValue
```

Remote tag references also support SuperTags. For example, a valid SuperTag remote reference is:

```
PLC1:"TankFarm\Tank1\Pump1RPM.RawValue"
```

## Importing SuperTags with the Bulk Import Utility

You can use the ArcestrA Bulk Import Utility to transform tag definitions from InTouch into an Application Server object structure. The ArcestrA Bulk Import Utility enables you to more efficiently migrate your InTouch tags to Application Server. In addition to standard InTouch tags, the ArcestrA Bulk Import Utility can also migrate SuperTags.

If you plan to migrate SuperTags from an InTouch application to Application Server, replace the backslash character (\) within your SuperTag reference to a supported ArcestrA character like an underscore (\_).

**Example:**

```
TankFarm_Tank1_Pump1RPM.RawValue
```

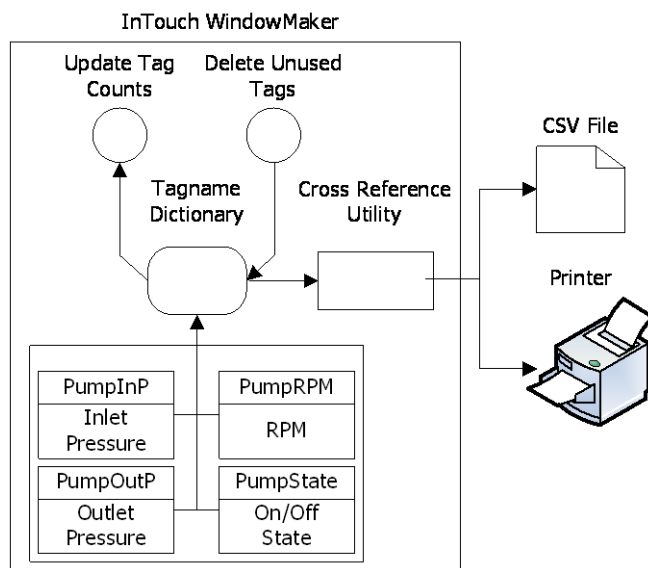
## Chapter 8

# Reducing Tag Usage

## About Reducing Tag Usage

The maximum number of tags you can use in an InTouch application is based upon your license. Your license prevents a running application's concurrent tag count from exceeding the maximum limit.

You can track tag usage in an InTouch application. The figure below shows how the Cross Reference Utility produces a tag usage report by analyzing the contents of the Tagname Dictionary.



You should maintain the minimum number of tags required to run your InTouch application. One way to minimize your tag count is to delete unused tags. You must update the tag count before deleting unused tags.

## InTouch and the Historian

The Historian is a real-time and historical database that stores plant process data. It combines the storage capacity of a relational database with the speed of a real-time system. As an enhancement to Microsoft SQL Server, the Historian acquires plant data at dramatically increased speeds. It also integrates plant data with event, summary, production, and historical data from the InTouch HMI and other related products.



The InTouch HMI can store historical tag data to remote historical repositories. In this case, you must share the InTouch HMI application directory and configure Historian to access the InTouch application. If you use the Historian to store InTouch HMI historical data, you must use the Distributed Name Manager from WindowMaker to specify a connection to the database. The InTouch HMI can show historical data stored in a Historian database in a Trend Wizard. You need to install SQL Server client components and browse the Historian computer to access historical tag data.

## Determining Tag Usage

The InTouch HMI maintains a use count for all tags defined in the local Tagname Dictionary. The tag count does not include internal system tags.

Remote tags are not defined in the Tagname Dictionary. InTouch increments the tag count for each reference within an application to a remote tag. For more information about how InTouch counts a remote tag reference, see *Determining Maximum Number of Remote Tags Based on Licensing* on page 130.

Before you delete unused tags, make sure that you complete the following tasks:

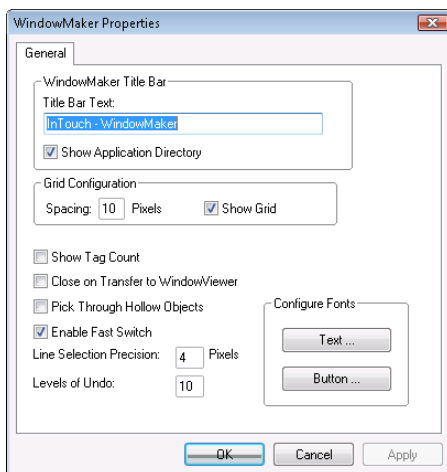
- Close WindowViewer.
- Update local tag counts and references to remote tags.
- Produce a Cross Reference tag report.
- Locate tag usage within an application with the Cross Reference Utility.
- Save your application in WindowMaker.

## Determining Tag Counts

You can update the count of local tags that are currently defined in an application’s Tagname Dictionary. You can also update the count of an application’s references to remote tags located on another node.

### To show the local tag count

1. Open an InTouch application in WindowMaker.
2. In the Tools view, expand the **Configure** list and select **WindowMaker**. The WindowMaker **Properties** dialog box appears.

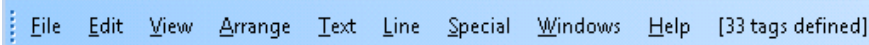


3. Select **Show Tag Count**.

When you select this option, the entire Tagname Dictionary must be read to update the displayed tag count. Updates to the Tagname Dictionary may take longer.

4. Click **OK**. A message appears that WindowMaker must be restarted before the configuration changes become effective.
5. Close WindowMaker.
6. Restart the application in WindowMaker.

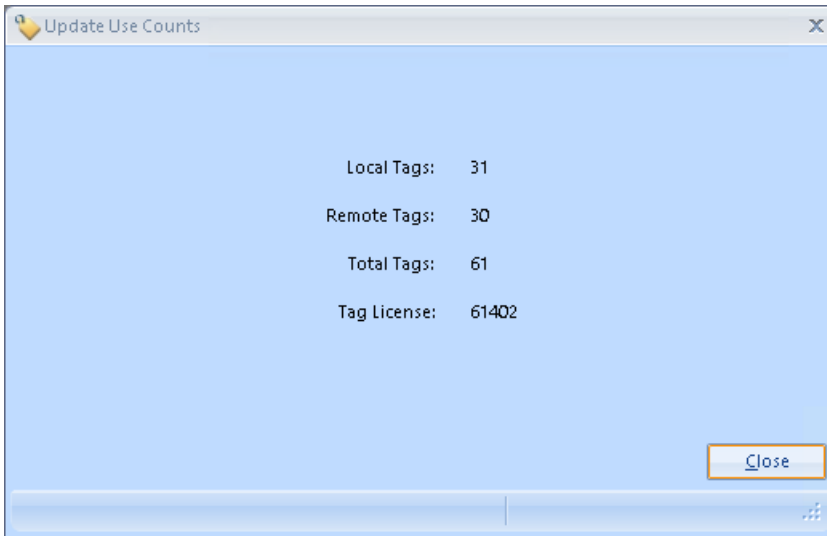
The number of local tags defined in the application’s Tagname Dictionary are shown at the right of the WindowMaker menu bar.



**To update the total tag count including remote references in an application**

1. Open an InTouch application in WindowMaker.
2. Close all open windows.
3. On the **Special** menu, click **Update Use Counts**.

After the calculations are complete, a dialog box shows the counts for the application’s local tags and references to remote tags.



The dialog box also includes a total tag count and maximum tag usage based upon the InTouch license.

## Determining Maximum Number of Remote Tags Based on Licensing

Your product license enforces the number of tags you can use with your InTouch applications. A license enforces not only the local tag count, but also the number of tags that reference tag sources located on remote nodes.

For more information about how an InTouch license enforces the number of remote reference tags that can be used in your applications, see *InTouch Licensing* on page 210.

## Locating Where Tags are Used

Using the Cross Reference Utility, you can produce an online report that shows tag usage within an InTouch application.

The Cross Reference Utility report shows local tags, remote tags, and SuperTags in:

- Animation links
- Wizards
- All types of InTouch scripts
- QuickFunctions
- Active controls
- Optional InTouch components such as SQL Access Manager and Recipe Manager.
- Industrial Graphics references

### To create a report with the Cross Reference Utility

1. Open an InTouch application in WindowMaker.
2. On the **Special** menu, click **Cross Reference**. The InTouch **Cross Reference** grid appears.

The Cross Reference grid lists all records related to local and remote tags defined in the current application's Tagname Dictionary.

## Understanding the Cross Reference Utility Report

In a grid format, the report lists the name of the reference type, type of tag reference, use, position on the window, window name, graphic name, full hierarchical path of the Industrial graphic having the tag reference, and the condition type if the tag is used in a script.

Name	Type	Use	Position	Window Name	Graphic Name	Hierarchical Name	Where
Mixer400_Inlet1_OIS	InTouch Tag						
Mixer400_Inlet1_Position	InTouch Tag						
Mixer400_Inlet2_OIS	InTouch Tag						
Mixer400_Inlet2_Position	InTouch Tag						
Mixer400_Level_PV	InTouch Tag						
Mixer400_MixingTime_SP	InTouch Tag						
Mixer400_Outlet_OIS	InTouch Tag						
Mixer400_Outlet_Position	InTouch Tag						
Mixer400_Pump1_PV	InTouch Tag						
Mixer400_Pump2_PV	InTouch Tag						
Mixer400_Temperature_PV	InTouch Tag						
Mixer400_Totalizer1_PV	InTouch Tag						
Mixer400_Totalizer2_PV	InTouch Tag						
MixerGroup_Inhibit	In-Alarm Inhibitor Tag						Mixers
MixerNo	Animation Link	At (130, 30) By (000,60)					KPI
MixerNo	Animation Link	At (40, 40) By (49,59)					KPI
Mixers	InTouch Tag						
MjFan	InTouch Tag						
None	Animation Link	At (600, 40) By (720,70)					History
None	Animation Link	At (605, 120) By (756,169)					History
None	Animation Link	At (610, 190) By (740,236)					History
Tag_Discrete1	InTouch Tag						
Tag_Discrete2	InTouch Tag						
Tag_Integer	InTouch Tag						
Tag_Message	InTouch Tag						
Tag_Reset	InTouch Tag						
UserLogOff	Industrial Graphic	At (795, 10) By (1031,90)			SecurityLogon	SecurityLogon	UserLogOff Condition
UserLogOff	Condition Script						UserLogOff On True
UserLogOff	Condition Script						UserLogOff On True
UserLogOn	Industrial Graphic	At (795, 10) By (1031,90)			SecurityLogon	SecurityLogon	UserLogOn Condition
UserLogOn	Condition Script						UserLogOn On True
UserLogOn	Condition Script						UserLogOn On True
WindowChoice	Industrial Graphic	At (130, 40) By (130,61)			Nav	Nav	WindowChoice
WindowChoice	Data Change Script						WindowChoice












For example:






Name	Type	Use	Position	Window Name	Graphic Name	Hierarchical Name	Where
NSelect	InTouch Tag	Key Script					F6 On Key Down
PLCSim.SP3	Object Attribute	Industrial Graphic	At (0,0) by (826,455)	Section5	SA_Meters2	SA_Meters.SA_Tank_Vessel.SA_Operating Range	

The status bar at the bottom of the report displays the:

- **No. of records:** Total number of records related to tags in the application. This value will change dynamically based on the applied filter.
- **Local Tags:** Total number of tags created on this node.
- **Remote Tags:** Total number of tags that reference remote tags.
- **Total Tags:** Total number of tags in the InTouch application.
- **Tag License:** Total number of tags allowed as per the license.

Icons appear in the InTouch Cross Reference Utility report to show status and usage.

Icon	Description
	The tag or SuperTag is defined in the application's Tagname Dictionary, but is not assigned to an object.
	The tag or SuperTag is used in an animation link, InTouch QuickScript or Industrial graphic.
	The Industrial graphic is referencing a tag or SuperTag
	The tag or SuperTag is assigned to an animation link.
	The tag or SuperTag is used in an Application script.
	Shown for all selected scripts. Double-click on the script name to view the script in read only mode.
	The tag or SuperTag is used in a Window script.
	The tag or SuperTag is used in a Data Change script.
	The tag or SuperTag is used in a Condition script.
	The tag or SuperTag is used in a Key script.
	The tag or SuperTag is used in a QuickFunction.

Icon	Description
	The tag or SuperTag is used in an ActiveX Event script.
	When cross-referencing by <b>Window</b> , this icon precedes the window name in which the displayed tag or SuperTag is used. Double-click on the icon to view all tags used in the window.
	Displayed tag or SuperTag is used in a SQL application.
	Displayed tag or SuperTag is used in a Recipe Manager application.
	The tag is used as an alarm inhibitor.

## Including Graphics from the Graphic Toolbox

- Click **Refresh** to view the latest tag information. If you edit the tag information or edit a symbol embedded on a window, while the Cross Reference window is open, then the Refresh button is enabled.
- Select the **Include all graphics from Graphic Toolbox** checkbox to view the tags that are not used in any Windows but present in the Graphic toolbox. By default, the checkbox is not selected.

**For example:** 10 tags are configured in the tag dictionary, 4 are referenced in graphics in the graphic toolbox (but not embedded on any windows), 4 tags are referenced on graphics placed on a window. By default, 4 tags (tags placed on a window) will be displayed. If the checkbox is selected, then 8 tags (4 from the window + 4 from the graphics) are displayed.

The graphics displayed are subject to the following conditions:

- Remote tag references within symbols, which are not used in InTouch windows will not be displayed.
- Object symbols, which are not used in InTouch windows, will not be displayed.
- If the checkbox is selected in the Cross Reference Utility, the condition will also be set for the other operations like Update Use Count and Delete Unused Tags.

## Cross Reference Tag Usage for InTouch Tags in ArchemstrA or Situational Awareness Library Symbols

The InTouch Cross Reference utility has been enhanced to include Industrial Graphics in an application's standard InTouch tag cross reference list. A cross reference search for Industrial Graphics can be conducted by the criteria shown below, which will be included as part of the Cross Reference utility's InTouch Tag Use.

- Custom property with reference to an InTouch tag
- Animation with direct reference to an InTouch tag
- Client script with direct reference to an InTouch tag
- Embedded symbols' reference to an InTouch tag

The Cross Reference Utility will also search Industrial graphics for attribute references, which are part of the Object Attribute. The exceptions to the tag usage report are:

1. Object Attribute references:
  - Object Attribute using relative reference (such as me.l1) will not be supported and will not be displayed in the cross reference utility tag usage report.
  - Reference using <InTouchViewApp instance name>.<Tagname> syntax will not be considered as InTouch Tag reference. This reference will be considered as Object Attribute.  
**Example:** InTouchViewApp1.Tag1
  - Any reference that is not in "InTouch:<Tag>" syntax will be considered as Object Attribute reference.
2. Client script references that are used as the script function parameters will not be considered as InTouch tag or object attribute reference.

**Example:** reference as string parameter  
SetCustomPropertyValue( "CP1", "InTouch:Tag1", false);

**Example:** reference as Attribute parameter  
SetBad(InTouch:Tag1);

3. Industrial Graphics referencing to an InTouch tag that is not defined in the current InTouch application will be displayed with Type as "Undefined".


**Example:** Object symbol  
ArchestrA object "UD\_001" has instance symbol "S1", "S2", and "S3".  
The instance symbol "UD\_001.S1" has reference to InTouch tag "Tag51".  
Embed UD\_001.S1 into UD\_001.S2;  
Embed UD\_001.S2 into UD\_001.S3;  
Embed UD\_001.S3 into InTouch window "Win1".  
Also embed UD\_001.S2 into window "Win1".

The instance symbol can be defined within the instance or in its derived template. The instance can be embedded using absolute or relative reference.

## Displaying Tag Usage in the Cross Reference Utility

You can modify the display of the cross reference tag usage report using the filtering, sorting and grouping options. Applying filters will update the number of records displayed. Sorting and grouping will only modifying the way the records are displayed on the grid.

### Refining the tag usage report using the filter option

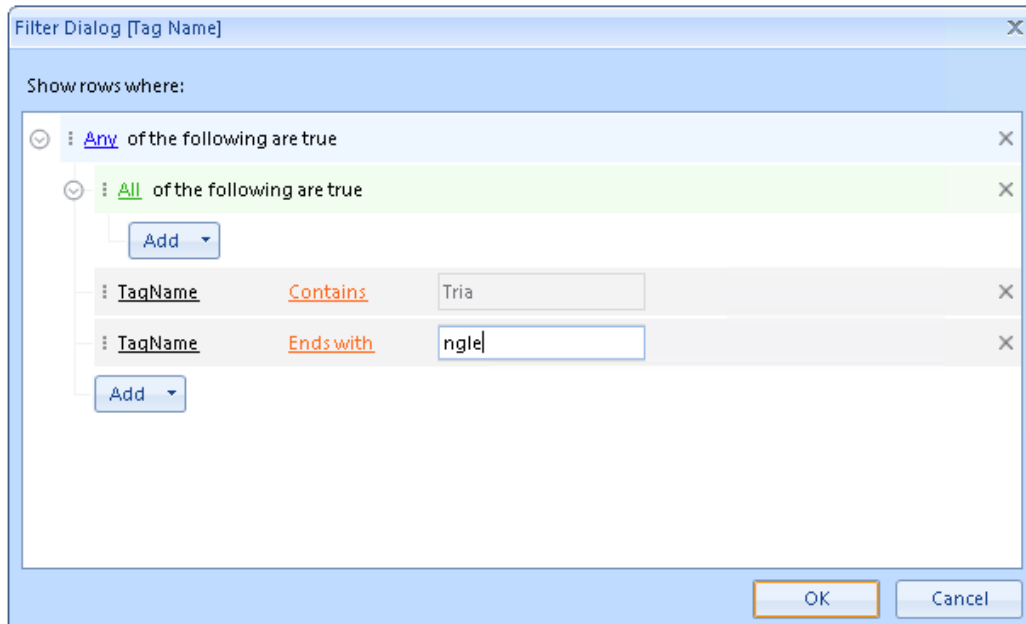
1. Use the Filter icon (  ) to select from a list of options like Contains, Does not contain, Starts with, Ends with, Equals, Not equal to, Is null, Is not null and Custom.
2. Enter the search term against the selected filter option.

The grid will display all records that match the filter option and the search term.

	Name
	Contains: Tria
	Triangle1
	Triangle10
	Triangle2
	Triangle2
	Triangle3
	Triangle4
	Triangle5
	Triangle6
	Triangle7
	Triangle8
	Triangle9
	Triangle1
	Triangle1
	Triangle1

### Using the custom filter option

The custom filter option allows you to add multiple tags and create a complex search expression.



### Sorting the tag usage report

Each column in the tag usage report can be sorted in ascending or descending order.

1. Click the column heading.

All records in the grid will be sorted based on the column. The direction of the arrow determines the order.

- Click the column heading again to change the order.

	Name	Type	Use
Contains:Tr		Contains:	Contains:
	\$DateString	InTouch Tag	
	\$TimeString	InTouch Tag	
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrend	InTouch Tag	Animation Link
	HistTrendPenScale	InTouch Tag	Animation Link
	Triangle1	InTouch Tag	Application Script
	Triangle10	InTouch Tag	Application Script
	Triangle2	InTouch Tag	Application Script
	Triangle2	InTouch Tag	Application Script
	Triangle3	InTouch Tag	Application Script
	Triangle4	InTouch Tag	Application Script
	Triangle5	InTouch Tag	Application Script

### Grouping the tag usage report

Organize the tag usage report in a pattern using a combination of columns.

- Select a column name and drop it to the empty space above the column names.  
A box with name of the column appears.
- Select another column name and drop it above or below the first box.  
The utility will automatically create a hierarchical relationship between the two columns.
- To create two columns at the same level drop the second column name on top of the first box.
- Sort individual columns in ascending or descending by clicking the column name box within the pattern.

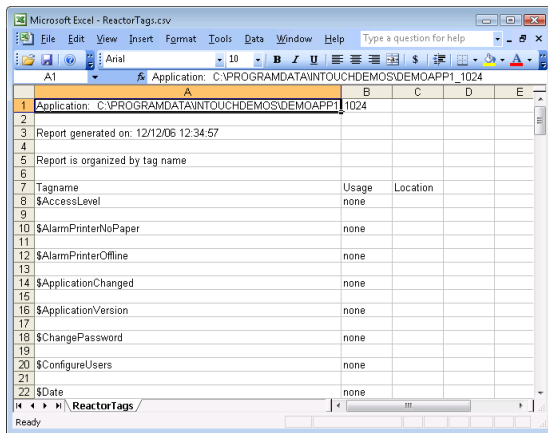
Group by:	Graphic Name	Type	Use	Name	Position	Window Name	Graphic Name	Hierarchical Name	Where
Contains:		Contains:	Contains:						
	Graphic Name								
	Graphic Name: ButtonArrowHoz								
	Graphic Name: Fan								
	Type, Use: InTouch Tag, Industrial Graphic								
	Name: Triangle2	InTouch Tag	Industrial Graphic	At (40, 33) By (190,100)	F4	Fan	Fan		
	Graphic Name: FanLight								

### Saving and Printing a Tag Cross-Reference List

You can save your cross reference tag listing to a file and view it with any application that supports the .csv file format.



The cross reference tag listing file lists all tags by name, how they are used in the application, and the name of the window where they are located. You can open the cross reference tag listing file with Excel or any other program that supports .csv files.

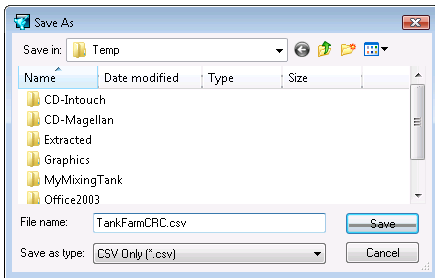


You can also print the contents of the Tagname Dictionary. Printing the contents of the Tagname Dictionary shows you database entries used in the application. You can specify the level of detail you want to see.

You can send this report to the printer or save it to a file.

**To save a cross-reference file**

1. In the InTouch **Cross Reference Utility** dialog box, click **Save As**. The **Save As** dialog box appears.



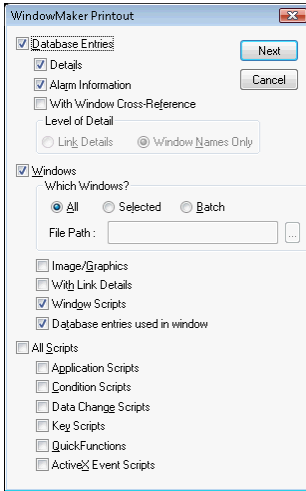
2. Specify a name and location for the file.
3. Click **Save**. The tag usage file is saved in the specified folder.

The tag usage report saved to a .csv file will depend on the following conditions:

- If no search conditions are selected then all records are saved to the .csv files. For example: If the total number of records in the cross reference grid is 1000 and on refining the search, the No. of records reduces to 320. If you use Save As... only the selected 320 records are saved to the .csv file.
- If you have applied a grouping pattern to your records, the .csv will be saved with that grouping pattern.

**To print the contents of the Tagname Dictionary**

1. Open an InTouch application in WindowMaker.
2. On the WindowMaker **File** menu, click **Print**. The **WindowMaker Printout** dialog box appears.



3. Select **Database Entries** if you want to print tag information from the Tagname Dictionary.

If you select **Database Entries**, the following options become active:

- Select **Details** to include the details of the tags in your report.
- Select **Alarm Information** to include the tag alarm information in your report.
- Select **With Window Cross-Reference** to print all Tagname Dictionary entries with window cross-references. If you select this option, specify the level of detail to print.

**Link Details** prints the location and animation link details where the tag is used.

**Window Names Only** prints only the name of the cross-referenced windows.

4. Click **Next**. The **Select Output Destination** dialog box appears.
5. Select the option to print the contents of the Tagname Dictionary or send the output to a text or .html file.
6. Click **Print**.

You can save your cross reference tag listing to a file and view it with any application that supports the .csv file format.

## Deleting Unused Tags

You can delete unused tags from an InTouch application after updating the use count. You can delete tags one at a time from the Tagname Dictionary, or you can delete several tags at one time.

The tag count is not automatically updated when a window containing tags is deleted from an application or tags are changed in link scripts. InTouch regards the tags as being in use and prevents them from being deleted.

---

**Note:** Deleting a single tag from the Tagname Dictionary: If the **Delete** button is disabled for a unused tag in an Industrial Graphic reference, run **Update Use Counts** to enable the Delete button.

---

The tag count must be updated to remove unused tags from the total before you can delete unused tags. For more information about updating tag counts, see *Determining Tag Counts* on page 129.

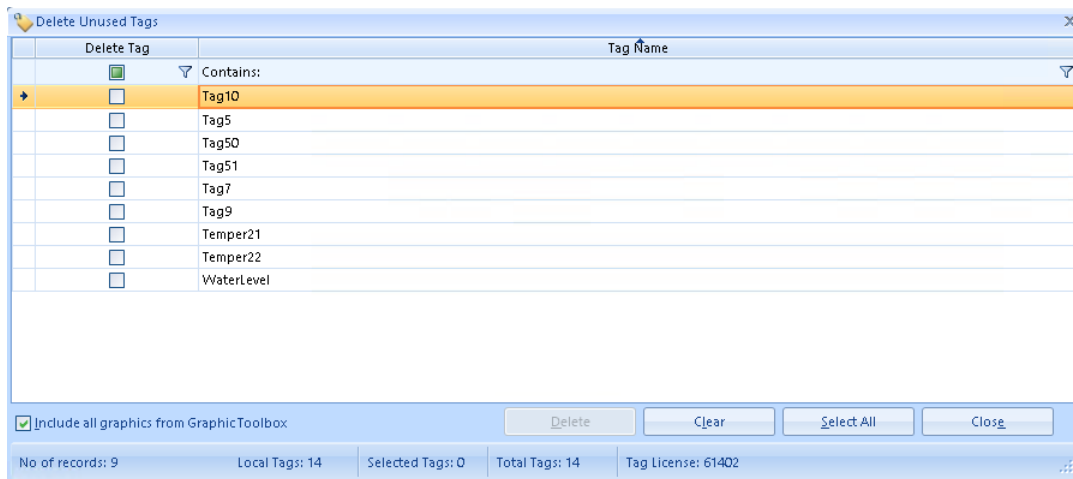
**Caution:** Tags that are only alarmed have no use count and can be accidentally deleted. To ensure that alarmed only tags are included in the use count, you need to use them in a window or QuickScript.


**To delete multiple unused tags**

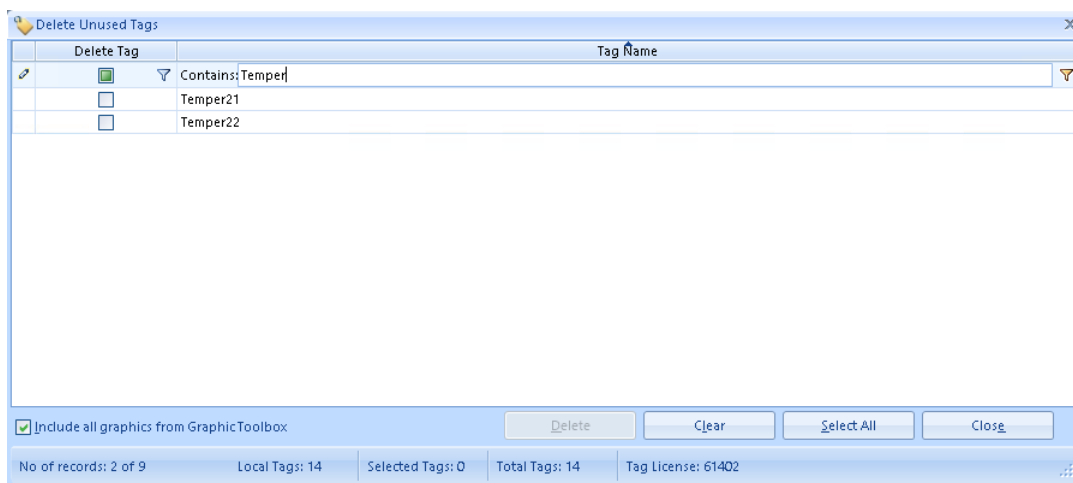
1. Close WindowViewer if it is running.
2. Open an InTouch application in WindowMaker.
3. On the **Special** menu, click **Delete Unused Tags**.

The **Delete Unused Tags** dialog box appears with a list of unused tags.

The status bar at the bottom of the dialog box lists the Number of records, Number of Local tags, Number of selected tags, Total number of tags and the Tag license count.



4. Use the filter (  ) option to refine the list of unused tags. Select the filter option and enter a search term. The tags matching the criteria are displayed.
5. Click the column headings to sort the columns.



6. Select the tags you want to delete. To select all tags for the current filter criteria, click **Select All**.

---

**Note:** The status bar displays the number of tags that have been selected (for example: No. of records: 1 of 14). The tag selection is maintained even if the column criteria is changed. The “Selected Tags” value will reflect the total number of tags that user has selected, independent of filter criteria. To view the entire list of selected tags, clear all the filter criteria.

---

7. Click **Clear**, to clear the tag selection for the current filter.
8. Click the **Include all graphics from Graphic Toolbox** checkbox to only see tags that are not used in any graphics or windows. These tags are safe to delete as they are not referenced in graphics or windows that contains graphics.

**For example:** 10 tags are configured in the tag dictionary, 4 are referenced in graphics in the graphic toolbox (but not embedded on any windows), 4 tags are referenced on graphics placed on windows. By default, 6 tags (2 unused tags + 4 tags from the graphics in the graphic toolbox not embedded on any window) will be displayed. If the checkbox is selected, then only the 2 unused tags are displayed.

9. Click **Delete**, to delete all tag selections independent of the filter criteria.

A confirmation message appears. The message will specify the number of tags that will be deleted.

10. Click **OK** to confirm.

## Chapter 9

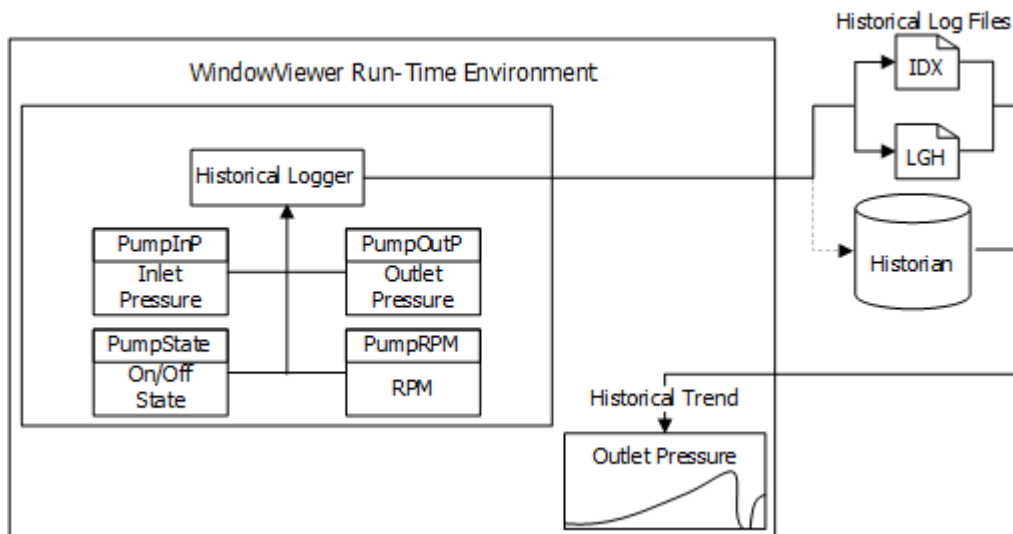
# Recording Tag Values

## About Recording Tag Values

While an InTouch application is running, its tag values can be logged and permanently saved. You can use this log data to create historical trend graphs that show some aspect of your plant processes over time.

**Note:** You should use Historian Server to save historical InTouch data for large applications, or when you want to create detailed reports. Refer to Historian Server documentation for details about configuring historical logging.

The following figure shows tag data from a pump saved to historical log files and/or Historian Server database. The InTouch Historical Logger writes a log entry each time a tag's value changes more than its specified log deadband range.



When storing data to historical log files, the InTouch HMI creates two log files. One file contains logged data stored in a proprietary format. The other file is an index to the data.

Names are assigned to the two log files with the following format:

`YYMMDD00.LGH` and `YYMMDD00.IDX`

where:

**YY** Last two digits of the year the log files are created

- MM** Two-digit number of the month the files are created (01-12)
- DD** Two-digit day of the month the files are created (01-31)
- 00** Constant value of 00 in the log file names

**A daily logging cycle begins and ends at midnight. The Historical Logger writes the last entries to the active log files at midnight and archives them. Two new files are created for the next day and data is logged to them.**

Log files are saved for a specified number of days. Log files that are older than the retention period are deleted. For more information about setting the number of days to retain log files, see *Configuring General Logging Properties - Historical Log File* on page 143.

## Configuring Historical Logging

You need to perform three main tasks to configure historical logging for an InTouch application:

- Configure tags for historical logging
- Configure general logging properties for the InTouch application
  - Configure properties for storing tag values in InTouch historical log files and/or
  - Configure properties for storing tag values in a Historian server
- Optionally, set the historical logging frequency

---

**Note:** You can configure logging to LGH files and/or the Historian server.

---

## Configuring Tags for Historical Logging

You select tags for historical logging from the Tagname Dictionary. When a selected tag's value changes, the Historical Logger determines if an entry should be written to the log based upon each tag's logging deadband and its current value.

If you change a tag from logged to not logged, the data associated with the tag is no longer saved to the log file. Logging resumes when logging is re-enabled. However, the historical trend shows a gap during the period when logging was disabled.

Changes to tag logging are ignored while WindowViewer is running the application. Logging changes to tags do not become effective until the running application is stopped and restarted.

You configure logging for each individual tag from the Tagname Dictionary.

### To enable historical logging for tags

1. If needed, stop WindowViewer from running the application.
2. Open the application with WindowMaker.
3. Open the Tagname Dictionary.
4. Select a tag from the Tagname Dictionary list whose data you want to log.

5. Select the **Log Data** check box.

The screenshot shows the 'Tagname Dictionary' dialog box with the 'Details' tab selected. The 'Log Data' checkbox is checked. The 'Tagname' field contains 'Tlog6', 'Type' is 'Memory Integer', and 'Local Tag' is unchecked. The 'Group' is '\$System' and 'Read/Write' is selected. The 'Comment' is 'AccessLevel'. 'Log Events' is checked with a priority of '999'. 'Retentive Value' and 'Retentive Parameters' are unchecked. The bottom section contains input boxes for 'Initial Value' (0), 'Min Value' (0), 'Deadband' (0), 'Eng Units', 'Max Value' (100), and 'Log Deadband' (0).

The **Tagname Dictionary** dialog box includes other tag attributes closely associated with logging:

- **Log Deadband** sets a engineering units threshold that must be exceeded before a tag’s value is written to the log file. Only new values outside of the deadband are written to the log file. Small value changes within the deadband range are ignored.
- The **Min EU** and **Max EU** properties scale clamped raw values within a range of engineering units. Minimum and maximum EU properties set the upper and lower boundaries of the scaled values.

The Min/Max engineering units determine the range boundary for log values shown in a trend. By default, an InTouch historical trend shows log data from 0-100 percent of the EU range.

6. Click **Save**.
7. Repeat these steps to enable logging for each tag whose data you want to log.
8. Click **Close** to close the Tagname Dictionary when you are done.

## Configuring General Logging Properties - Historical Log File

You can set general logging properties that apply to the selected application.

### To configure historical logging

1. If necessary, close the InTouch application running in WindowViewer.
2. Open WindowMaker.

- On the Tools view, expand **Configure** and select **Historical Logging**. The **Historical Logging Properties** dialog box appears.

Historical Logging Properties

For more information about setting up trend printing, see *Printing a Trend at Run Time* on page 190.

- Select the **Enable Historical Logging** check box.
- In the **Keep Log Files for** box, type the number of days prior to the current day to retain log files.

Log files are kept for the current day and the number of days within the specified retention period. Log files that are older than the retention period are deleted. Setting the value to 0 retains all log files indefinitely.

**Example:**

Set the retention period to five days and began logging on the first day of the month. On the seventh day of the month, the log files are retained from the five previous days and the current day (02-07). The log files created on the first day of the month are deleted.

Consider disk space usage when you set the number of days to save logging data. Historical logging stops if your hard disk runs out of free space. You must free disk space to resume logging.



6. Select the location of the folder to save log files.

The **Historical Logging Properties** dialog box includes two options to set the folder location to store the log files.

---

**Note:** The folder path and the name of the file to store log data can be a maximum of 55 characters.

---

Select **Store Log Files in Application Directory** to save the log files in the same folder as the InTouch application creating the logged data.

Select **Store Log Files in specific Directory** to specify another folder to store log files. You can specify the folder to store log files as:

- Windows folder path such as C:\History Log Files
- Universal Naming Convention (UNC) path such as \\node\share\directory.

If you are saving log files to a distributed node, you must specify the directory as a UNC path.

If configured to write historical data to the master application node's Application Directory, all NAD nodes try to write historical data to the master application. To avoid this, configure historical data on each NAD node to write to a local directory, not the master application node.

7. In the **Name of Logging Node** box, type the node name of the computer running the InTouch application creating log data.
8. Click **OK** to save your settings.

The logging configuration changes become effective immediately. Logging begins the next time you run the application.

## Configuring General Logging Properties - Storage to Historian

You can set general logging properties that apply to the selected application.

### To configure storage to Historian

1. If necessary, close the InTouch application running in WindowViewer.
2. Open WindowMaker.

- On the Tools view, expand **Configure** and select **Historical Logging**. The **Historical Logging Properties** dialog box appears.

Historical Logging Properties

For more information about setting up trend printing, see *Printing a Trend at Run Time* on page 190.

- Select the **Enable Storage to Historian** check box.
- In the **Historian node name** box, type the node name of the computer running the Historian server. You can also specify the IP address and if the server is installed on the same machine you can use 'localhost'. Only the following special characters are allowed; full stop (.), underscore (\_) and hyphen (-).
- In the **History store forward directory** box, type the path to the local folder location where the files related to store forward are stored. The files will allow the historical data to stored temporarily if the connection to the Historian server is lost. After the connection is established, the Historian server will sync with the files from this store forward directory and preserve all information.
- Click **Advanced Settings...** to specify settings related to the Historian connection.

- a. Under the **Connection** section, you can specify the **TCP Port**. The TCP port on the Historian Server node to which history data will be sent. The TCP port is configured when the Historian Server is installed. The default port is 32568.
  - b. Under the **Bandwidth Optimization** section, you can select the **Enable compression** checkbox. Once selected you can specify:
    - **Throttling network bandwidth**: Specifies limit of bandwidth usage, in kbps, for network communication used by HCAL when communicating with the Historian. A value of 0 disables this feature (default). For more information on estimating your bandwidth needs, see the performance and sizing recommendations for the Historian Server in the System Platform Installation Guide. The allowed values are 0 kbps to 65535 kbps.
    - **Wait to send incomplete packets**: Specifies the maximum time, in milliseconds, for keeping a partially-filled Historian Client Access Layer (HCAL) buffer before sending it to the Historian Server. If the buffer is full, then it is sent immediately, regardless of any value configured in this field. If you have fast-changing data, this setting is irrelevant. If you have slow-changing data and limited network bandwidth, you might want to increase this value so that you have less "chatter" on the network. The allowed values are 1000 milliseconds to 30000 milliseconds.
  - c. Under the **Data Management** section, you can specify:
    - **Pre-processing buffer size**: The total size, in MB, of all buffers used by Historian Client Access Layer (HCAL). The default and minimum value is 8. If you have very high data bursts, then you should increase this value. If this is too low, data loss occurs and error messages related to buffer overflows appear in the ArchestrA Logger. Increase this value in increments of 10 MB. The allowed values are 8 MB to 65535 MB
    - **Store forward threshold**: The size, in MB, of free space to reserve on the HCAL store-and-forward disk. The space designated will not be used during store-and-forward. This value cannot be a negative number. The allowed values are 0 MB to 65535 MB
    - **Store forward minimum duration**: The minimum duration, in seconds, for HCAL to function in store-and-forward mode. HCAL will function in store-and-forward mode for this length of time even if the condition that caused HCAL to function in store-and-forward mode no longer exists. The allowed values are 30 seconds to 3600 seconds.
    - **Select the Reconnect as soon as possible and do not mark disconnects checkbox**: Specifies how trends appear during communication disconnects between Application Server and Historian. It does not affect how trends of history data appear after communications have been restored. If TRUE, no gap will appear in client-side trends for the disconnect interval. While disconnected, the interval is filled in with the last-received value before the disconnect. If FALSE, a gap will appear in client-side trends for the disconnect interval. NULL values are injected on disconnect to create the gap. In both cases, after reconnect, the interval will be filled in with store-and-forward data.
  - d. Click **OK**.
8. Select **Log Alarms and Events** to enable logging alarms and events. Specify the alarm query in the **Alarm Query** text box.

---

**Note:** If different InTouch applications store tag data to the same Historian server and use the same tag name, the InTouch applications will not detect such a scenario and may cause the Historian data to overlap. Use a unique prefix or suffix to differentiate between applications. For more information, see *Configuring the Affix String* on page 148.

---

9. Click **OK** to save your settings.

The logging configuration changes become effective immediately. Logging begins the next time you run the application.

## Configuring the Affix String

In scenarios when the same application is used on multiple nodes and connected to the same Historian Server, you can use a unique suffix or prefix to differentiate between tags from different nodes. You can configure the string via WindowMaker or by manually editing the 'dhistcfg.ini' file on each node.

1. Open WindowMaker.
2. On the **Tools** view, expand **Configure** and select **Historical Logging**. The Historical Logging Properties dialog box appears.
3. Select the **Enable Storage to Historian** check box.
4. Select **Always affix unique string** checkbox.
5. Choose between **Prefix unique string** and **Suffix unique string**.
6. Enter the string in the **Unique string** field. The string can be a maximum of 6 characters.
7. Click **OK**.

### Manually editing the affix string in the .ini file

You can manually update the affix string by editing the dhistcfg.ini file available within the application folder on each node.

1. Navigate to the application folder.
2. Edit the dhistcfg.ini file.

Example1: Assign a prefix unique string of "aa"

```
szHistorianNode=<MachineName>
bStorageLoggingEnabled=1
bAffixEnabled=1
bPrefixEnabled=1
bSuffixEnabled=0
szHistUniqueString=aa
```

Example2: Assign a suffix unique string of "xx"

```
szHistorianNode=<MachineName>
bStorageLoggingEnabled=1
bAffixEnabled=1
bPrefixEnabled=0
bSuffixEnabled=1
szHistUniqueString=xx
```

3. Save and close the .ini file.

Any changes will be effective after WindowViewer is restarted.

## Controlling Historical Logging Frequency

You can optionally specify that entries are logged based upon two conditions:

- An immediate log entry is written whenever a tag value changes by an engineering unit value greater than its log deadband value.
- The current values of all logged tags are written at a fixed interval. Entries for all logged tags are written regardless of their current values. The default fixed interval is 60 minutes.

You can accept the default fixed logging interval or add two parameters to the intouch.ini file to change the interval.

- *ForceLogging*

*ForceLogging* specifies the length of the fixed logging interval in minutes. *ForceLogging* can be set to a value between 5 and 120 minutes. The default is *ForceLogging=60*.

- *ForceLogCurrentValue*

*ForceLogCurrentValue* specifies that, at the fixed logging interval, the current tag value is logged, even if it is still within the log deadband of the last logged value. When set to 0, the last logged value is logged again. The default is *ForceLogCurrentValue=0*.

The following example shows an example of the intouch.ini file that includes the two logging parameters.

```
WinFullScreen=1
WinWidth=808
AlarmBufferSize=5000
ForceLogging=5
ForceLogCurrentValue=1
```

In this example, tag values are written to the Historical Log file at five minute intervals.

### To modify the historical logging frequency

1. Close WindowMaker and WindowViewer.
2. Locate the intouch.ini file in the same folder as the InTouch application.
3. Edit the intouch.ini file.
4. Insert a *ForceLogging* statement with a value between 5 and 120.
5. Insert a *ForceLogCurrentValue=1* statement.
6. Save your changes and close the intouch.ini file.
7. Restart WindowViewer.

## Starting and Stopping Historical Logging at Run Time

When an application is running, you can manually stop and restart historical logging with commands from the WindowViewer **Special** menu.

- The **Stop Historical Logging** command stops logging for the duration of the current application session. Logging remains stopped for the current session until it is manually started again.
- The **Restart Historical Logging** command restarts logging after it is manually stopped with the **Stop Historical Logging** option.

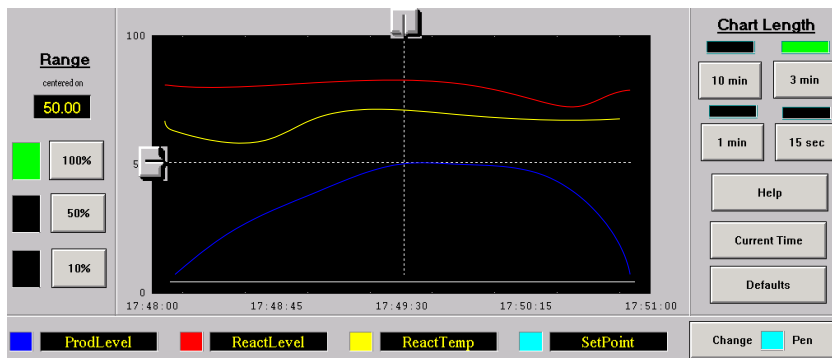
You can also add a button to your application and write a QuickScript that includes the **\$HistoricalLogging** system tag to start and stop historical logging. Logging starts when **\$HistoricalLogging** is assigned a value of 1. Logging stops when **\$HistoricalLogging** is assigned a value of 0. For more information about the **\$HistoricalLogging** system tag, see *System Tags* on page 32.

## Chapter 10

# Trending Tag Data

## About Trending Tag Data

You can create trends that graphically show data collected from an InTouch application. WindowMaker includes a set of utilities and wizards that enable you to create historical and real-time trends. The following figure shows an example of an InTouch Average/Scatter trend.



You can also use a set of trend controls. Using these controls, you can select the data shown in a trend and how data appears in the trend.

You can configure real-time and historical trends. Both trend types include configuration options to set a trend's data collection interval and visual appearance.

## Types of InTouch Trends

A historical trend shows log data collected from the past and stored in InTouch data repositories.

Using a distributed history system, you can retrieve historical data from any InTouch historical log file located on an accessible network node. The distributed history system extends the retrieval capabilities of historical trends to include remote log databases.

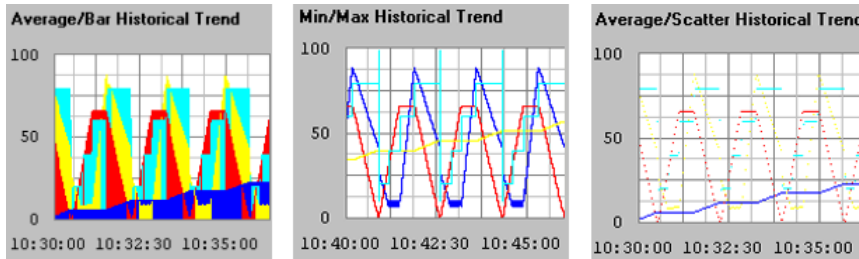
Real-time trends update continuously to show data as it occurs over relatively short periods. You can use WindowMaker's Real-time Trend tool to create a trend object in a window. If the optional 16-Pen Trend tool is installed, you can also create real-time trends that show data from up to 16 tags.

## Understanding Historical Trends

Historical trends show a contiguous segment of data from the past. Unlike real-time trends, historical trends are updated only by a script or an operator action.

A historical trend shows a graphical representation of data from a maximum of eight tags. You assign the data that appears in a historical trend by assigning a tag to a trend pen.

The figure below shows the three types of InTouch historical trends.



- The Average/Bar historical trend shows the average value of a data point during the time intervals in bar form.
- The Min/Max historical trend shows the changes in the percentage of engineering units scale as a vertical line over the time span. The emphasis is on time flow and rate-of-change, rather than amount of change.
- The Average/Scatter historical trend shows the average value of the data points over each trend time interval.

You can create graphical sliders called scooters to access the details of trend data based on the scooter's current position within a trend. For example, when the operator positions the scooter over an area on the trend that has visible data, the time and values at that location for all database values being trended are shown.

You can also create buttons or sliders to zoom in and out between the scooters or to data, such as the maximum to minimum value. Average and standard deviation can be shown for the complete chart or for the area between the scooters.

Historical trends can also be scrolled by any amount of time. You can create custom scales and link them to the **.MinEU** and **.MaxEU** dotfields to create a trend that shows the full range of data set by its engineering unit.

## Understanding Real-Time Trends

A real-time trend shows data from an InTouch application that is currently running. Real-time trends are continuously updated. Real-time trends plot current data values associated with up to four local tags or expressions.

You can:

- Create a real-time trend
- Select the tags for a trend
- Specify the time span and update interval of a trend
- Configure the display options of a trend



## Showing Saved Tag Values in a Historical Trend

You can create historical trends by using any of the following WindowMaker utilities:

- Historical Trend tool
- Historical Trend Wizard
- 16-Pen Trend Wizard (Optional)

In addition, you can incorporate a ActiveFactory trend to show InTouch historical trend data saved to a IndustrialSQL ServerArchestrA Bulk Import Utility database.

## Using Historical Trend Objects

You can create and configure a trend with the WindowMaker Historical Trend tool. You can

- Create a historical trend
- Select the tags for a trend
- Specify the time span and update interval of a trend
- Configure the display options of a trend

## Creating a Historical Trend

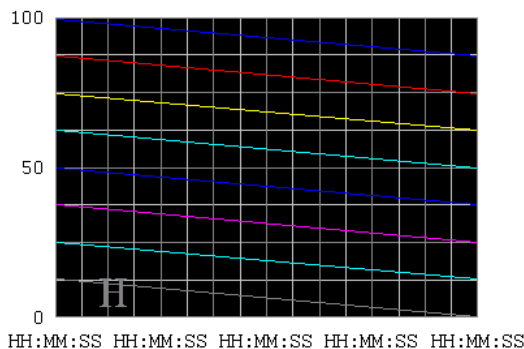
You can use the Historical Trend tool to create a trend object in a window. The first time you create a historical trend object, the InTouch default configuration settings are used.

After you configure a historical trend, WindowMaker uses the most recent configuration values as the initial settings for a new trend object.

You can draw the trend chart to any size within the borders of the window.

### To create a historical trend

1. Open the window in WindowMaker in which you want to place a historical trend.
2. Click the **Historical Trend** button from the Drawing Toolbar.
3. Move the mouse over the window area where you want to place the historical trend. Drag the mouse diagonally to create a rectangle the desired size of the trend. The Historical Trend object appears in the window.



4. If needed, adjust the height and width of the trend with its object handles.

## Configuring Which Tags to Show From a Historical Trend

A historical trend pen creates a graphical representation of logged data from a specified period. You assign trend pens to the tags that collect historical data.

A historical trend supports up to eight pens.

---

**Note:** WindowViewer must be closed. Otherwise, the Pen boxes cannot be selected.

---

You can select tags from remote history providers, if the providers are configured. For information about setting up a remote history provider, see Distributing Applications in the InTouch® HMI Application Management and Extension Guide.

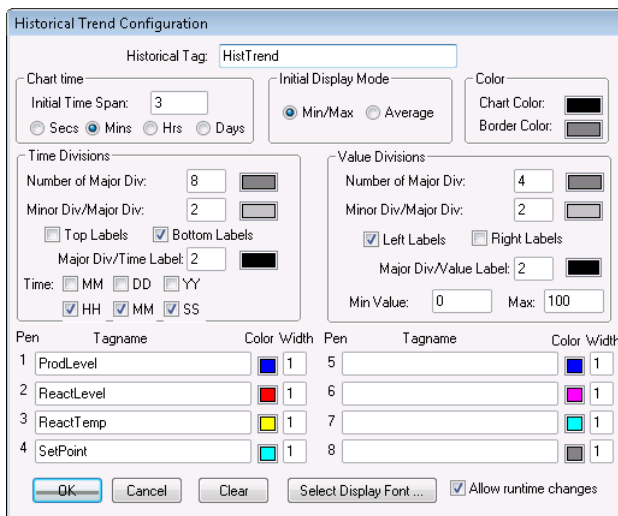
---

**Note:** You can also configure a IndustrialSQL Server history provider to visualize historical trend data. For more versatility and other charting options, use the ActiveFactory trend tools to create trends with InTouch historical data saved to a IndustrialSQL Server database.

---

### To configure which tags to display from a historical trend

1. Double-click the trend object in the window. The **Historical Trend Configuration** dialog box appears.



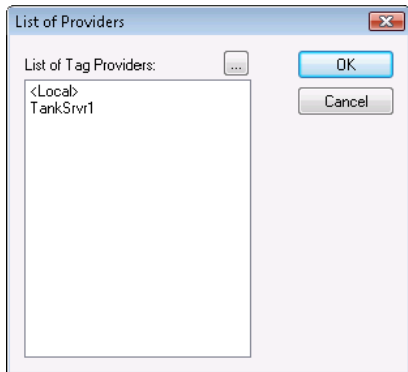
2. In the **Historical Tag box**, type the tag that you want to use for the trend.

The tag must be defined as a Hist Trend type. You must assign a different Hist Trend tag to each historical trend in an InTouch application.

If the tag you enter is not currently defined in the Tagname Dictionary, a dialog box appears and asks if you want to create a tag. If you select **OK** to define a tag, the **Tagname Dictionary** dialog box appears.

3. In the **Tagname** area, specify the name of an existing local or remote tag in one or more **Pen** boxes.
4. To assign an existing local or remote tag directly, click in a **Pen** box and type the name of the tag.
5. To browse to the tag to assign:

- a. Double-click in a **Pen** box. The **List of Providers** dialog box appears.



- b. Select the tag provider you want to use for the pen.
  - c. Click **OK** to show a dialog box listing the tags for the selected provider.
  - d. Double-click on a tag from the list to select it.
6. Double-click the color box next to each pen assigned a tag to show the color palette. Click the color for the pen.
  7. In the **Width** box, type the line width in pixels for each pen shown in the trend.
  8. Repeat steps 3 to 7 for each tag that you want to assign to a historical trend pen.
  9. If needed, select the **Allow runtime changes** check box to allow an operator to configure a historical trend while the application is running.

For more information about updating a historical trend during run time, see *Changing the Trend Configuration at Run Time* on page 157.

## Configuring the Time Span of a Historical Trend

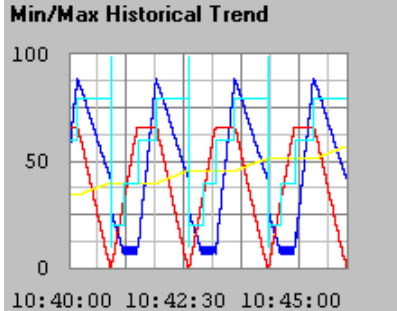
You can configure the time span of a historical trend.

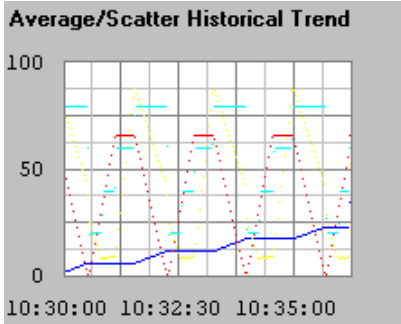
### To configure the time span of a historical trend

1. If needed, double-click on the trend object to show the **Historical Trend Configuration** dialog box.
2. In the **Chart Time** area, type the length of time in **Initial Time Span** that you want to appear on the horizontal x-axis of the trend.
3. Select the time unit of measure: Seconds (Secs), Minutes (Mins), Hours (Hrs), or Days (Days).

For example, if you enter 8 in **Initial Time Span** and then select **Hrs**, the time span shown on the trend is 8 hours.

4. In the **Initial Display Mode** area, select the type of historical trend that appears when WindowViewer initially shows the window containing the trend.

Initial Display Mode	Description
Min/Max	<p>The chart shows changes in the percentage of engineering units scale as a vertical line over the time span.</p> 

Average	<p>Each pixel within a trend time segment shows the average value of a tag over the period of time within the segment.</p> 
---------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5. Go to *Configuring Historical Trend Display Options* on page 156 to configure the visual appearance of a historical trend.

## Configuring Historical Trend Display Options

You can configure the visual appearance of a historical trend.

### To configure historical trend display options

1. If needed, double-click the trend object. The **Historical Trend Configuration** dialog box appears.
2. Set the color options. Do the following:
  - In the **Color** area, click the **Chart Color** box to open the color palette.
  - Click a color in the palette as the background for the trend.  
White is the default background color. Any other background color increases the time needed to print a trend.
  - Select **Border Color** to open the color palette.
  - Click a color in the palette as the border color of the trend.
3. Set the time divisions options. Do the following:
  - In the **Time Divisions** area, type the number of major trend time divisions in **Number of Major Div.**

The major time divisions appear on the horizontal time axis of the trend. The maximum time between major time divisions is 65536 seconds, or 18 hours, 12 minutes, 16 seconds.

- Select the color for the major division lines.
- From **Minor Div/Major Div**, type the number of minor time divisions within each major time division.

The number of minor time divisions should be evenly divisible within a major division. For example, if the major division is set to 60 seconds, entering a value of 2 in **Minor Div/Major Div** sets the minor time division to 30 seconds.

- Select the color for the minor division lines.
- Select **Top Labels** or **Bottom Labels** to specify the placement of time labels on the trend.
- If you are using time labels, type the number of major time division lines per time label in the **Major Div/Time Label** box.
- Select the color of the time division labels.
- Select the time units shown as the label of the major time division.

Months (MM)	Hours (HH)
Days (DD)	Minutes (MM)
Years (YY)	Seconds (SS)

4. In the **Value Divisions** area, configure the appearance of the vertical axis of the trend.

**Value Divisions** options are configured the same way as the **Time Divisions** options. The vertical axis specifies the range of data values that appear in the trend based upon engineering units for all tags.

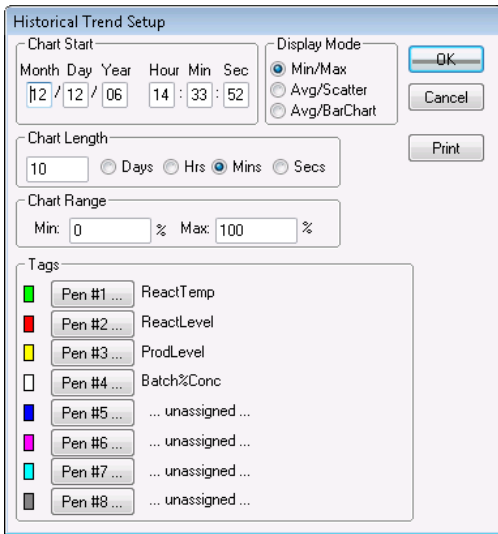
5. Click **OK** to save your configuration changes and close the **Historical Trend Configuration** dialog box.

## Changing the Trend Configuration at Run Time

If you select the **Allow runtime changes** option when you configure your historical trend, operators can configure a historical trend while it is running. Operators configure the trend from a dialog box that appears after selecting the trend from a displayed window.

**To configure a historical trend during run time**

1. Click on the historical trend while it is running. The **Historical Trend Setup** dialog box appears.



2. In the **Chart Start** area, type the starting date and time of the historical trend data collection interval.
3. In the **Display Mode** area, select the type of historical trend chart.

The trend display mode affects performance. The primary factor that determines trend performance is the length of the lines shown in a trend. The longer the lines, the longer it takes to generate the trend.

Line width also affects performance. Wide lines take significantly longer to draw. Min/Max or Average/Scatter trends can be created more quickly than an Average/Bar Chart.

4. In the **Chart Length** area, type the duration to be displayed on the trend and then select the unit of measure.

For example, if you enter 2 and select **Hrs**, the trend duration is 2 hours.

5. In the **Chart Range** area, type the percentage of engineering unit range shown on the vertical axis of the historical trend.

The scale of the trend is a segment of the trended tag’s engineering unit range defined by a percentage range. The values range from 0 to 100. For example, if you want to trend the variance of the selected tags from 40 to 60 percent of their engineering unit range, type 40 and 60 in the **Min** and **Max** range boxes, respectively.

6. In the **Tags** area, click a pen number to assign a tag.

The **Select Tags** dialog box appears with a list of tags that can be assigned to the historical trend pen.

7. Use the following statement in a QuickScript or button to allow the operator to update the chart:

```
Hist_TrendTag.UpdateTrend = 1;
```

8. Use any of the following functions in a QuickScript or on a button:

```
HTUpdateToCurrentTime(Hist_Tag);
```

```
HTScrollLeft(Hist_Tag,Percent);
```

```
HTScrollRight(Hist_Tag,Percent);
```

```
HTZoomIn(Hist_Tag,LockString);
```

```
HTZoomOut(Hist_Tag,LockString);
```

```
HTSetPenName(Hist_Tag,PenNum,Tagname);
```

For more information about using scripts containing trend functions, see *Controlling a Historical Trend Wizard Using Scripts* on page 174.

9. Change any of the following trend tag dotfields:

**.ChartStart**

**.ChartLength**

**.MaxRange**

**.MinRange**

**.Pen1-.Pen8**

For more information about using dotfields with historical trends, see *Controlling a Historical Trend Using Dotfields* on page 159.

## Controlling a Historical Trend Using Dotfields

You can use dotfields to manage historical trends during run time.

### .DisplayMode Dotfield

The **.DisplayMode** dotfield specifies the trend format used to display a tag's values.

#### Category

Historical

#### Usage

```
tag_name.DisplayMode
```

#### Parameter

*tag\_name*

Any Hist Trend tag.

#### Data Type

Analog (read/write).

#### Valid Values

1 = Shows the min/max value that occurred in each sample period (default).

2 = Shows the average value of each sample period in a scatter historical trend.

3 = Shows the average of each sample period in a bar chart historical trend.

#### Example

This statement specifies that the values in the historical trend represented by the "HistTrend\_Tag" are formatted as a bar chart historical trend.

```
HistTrend_Tag.DisplayMode=3;
```

### See Also

.ChartLength, .ChartStart

## .MinRange Dotfield

The **.MinRange** dotfield specifies the minimum percentage of the tag's engineering unit range to show for each tag in a Historical trend.

### Category

Historical.

### Usage

```
tag_name.MinRange
```

### Parameter

*tag\_name*

Any Hist Trend tag.

### Remarks

A historical trend can show several types of tags at the same time. Specifying the minimum and maximum boundaries of the value range in engineering units is difficult because different types of tags can have different engineering ranges. Therefore, the minimum and maximum range values are expressed as a percentage of the engineering range of each tag. This way, regardless of the tag's true engineering range, the historical trend shows the indicated percentage of that tag's particular engineering range.

### Data Type

Real (read/write).

### Valid Values

The limits for the .MaxRange and .MinRange dotfields are from 0 to 100. .MinRange is always less than .MaxRange. If you assign a value less than 0 or greater than 100 to either of these dotfields, the value is clamped at 0 or 100. If .MinRange is greater than or equal to .MaxRange, the trend does not show any data.

### Example

This example dotfield statement sets the minimum percentage range of the historical trend to 25 percent of the possible engineering unit range of a Hist Trend tag.

```
HistTrend.MinRange=25
```

### See Also

.ChartStart, .ChartLength, .DisplayMode, .EngUnits, .MinEU, .MaxEU, .MaxRange, .MinRaw, .MaxRaw, .RawValue

## .MaxRange Dotfield

The **.MaxRange** dotfield specifies the maximum percentage of the tag's engineering unit range to show for each tag in a Historical trend.



### Category

Historical.

### Usage

```
tag_name.MaxRange
```

### Parameter

*tag\_name*  
Any Hist Trend tag.

### Remarks

A historical trend can show many types of tags simultaneously. Specifying the minimum and maximum boundaries of the range in engineering units can be difficult because tags can have different engineering ranges. Therefore, the minimum and maximum range values are expressed as a percentage of the engineering range of each tag. This way, regardless of the tag's true engineering range, the historical trend shows the indicated percentage of that tag's engineering range.

### Data Type

Real (read/write).

### Valid Values

The limits for .MaxRange and .MinRange dotfields are from 0 to 100. .MinRange is always less than .MaxRange. If you assign a value less than 0 or greater than 100 to either of these dotfields, the value is clamped at 0 or 100. If .MinRange is greater than or equal to .MaxRange, the trend does not show any data.

### Example

This example dotfield statement sets the maximum percentage range of the historical trend to 75 percent of the possible engineering unit range of a Hist Trend tag.

```
HistTrend.MaxRange=75
```

### See Also

.ChartStart, .ChartLength, .DisplayMode, .EngUnits, .MinEU, .MaxEU, .MinRange, .MinRaw, .MaxRaw, .RawValue

## .UpdateCount Dotfield

The **.UpdateCount** dotfield increments a count each time a historical trend is updated. The **.UpdateCount** dotfield can be used as a trigger for further functions.

### Category

Historical.

### Usage

```
HistTrendTag.UpdateCount
```

### Parameter

*HistTrendTag*  
HistTrend tag assigned the name of the trend.

### Data Type

Integer (read-only).

### Valid Values

Any positive integer.

### Example

This example uses the **HTGetValueAtScooter()** function to retrieve the value of Pen1 at the right scooter's current position. A change to any function argument causes the function to be re-evaluated. When the update completes and the value of **.UpdateCount** is incremented, this statement is re-evaluated.

```
MyRealTag=HTGetValueAtScooter( MyHistTrendTag,MyHistTrendTag.UpdateCount, 2,  
MyHistTrendTag.ScooterPosRight, 1, "PenValue");
```

### See Also

.UpdateInProgress, .UpdateTrend

## .UpdateInProgress Dotfield

The **.UpdateInProgress** dotfield indicates the current status of a historical trend update operation. The value of the dotfield is set to 1 if a historical retrieval is in progress; otherwise the dotfield is set to 0.

### Category

Historical.

### Usage

```
HistTrendTag.UpdateInProgress
```

### Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.

### Remarks

Whenever new data is requested from the historical trend, this dotfield's value is set to 1. After the process completes, **.UpdateInProgress** is reset to 0. **.UpdateInProgress** can be used in functions related to historical trends.

If the operator scrolls the trend to a period outside the currently shown period, it can take some time to retrieve the historical data. The **.UpdateInProgress dotfield** provides a way to alert the operator that the requested data is being retrieved. Without feedback, the operator may not be aware that the trend is being updated.

### Data Type

Discrete (read-only).

### Value Values

0 = No update in progress

1 = Update in progress

### Example

The **.UpdateInProgress** dotfield is typically used as the expression in a visibility link on a text object near or on the scroll buttons of a Historical Trend. You can use the **.UpdateInProgress** dotfield to show "Busy" on the window when the data is being retrieved with the following message value display animation link:

```
DText(HistTrend1.UpdateInProgress, "Busy", "Ready")
```

### See Also

.UpdateCount, .UpdateTrend

## .UpdateTrend Dotfield

The **.UpdateTrend** dotfield triggers an update to a historical trend. Using the **.UpdateTrend** dotfield in a button action script, the operator can manually update the trend during run time.

### Category

Historical.

### Usage

```
HistTrendTag.UpdateTrend
```

### Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.

### Remarks

Historical trends do not automatically update. A change must be made to either the **.ChartStart** or the **.ChartLength** dotfields to update the chart and show the current values for the specified tags. By using this dotfield in a button action script, the operator can update the chart during run time. You can also use this dotfield in a QuickScript if other dotfields associated with the historical trend are going to be changed.

You should only set the **.UpdateTrend** dotfield to a value of 1.

### Data Type

Discrete (write only).

### Valid Values

1

### Example

This example triggers the historical trend associated with the **MyHistTrendTag** tag to update with the current values of all parameters.

```
MyHistTrendTag.UpdateTrend=1;
```

## .ChartLength Dotfield

The **.ChartLength** dotfield specifies the length of time shown in a Historical trend.

### Category

Historical.

**Usage**

```
HistTrendTag.ChartLength
```

**Parameter**

*HistTrendTag*

HistTrend tag assigned the name of the trend.

**Remarks**

The value assigned to .ChartLength specifies the length of the chart in seconds. The length is defined as the amount of time currently shown on the Historical Trend Chart. More specifically, the calculation retrieved as the Chart Length from a Historical Trend Chart is:

```
ChartLength=(Date/Time Stamp on Right-Hand Side of Chart) - (Date/Time Stamp on Left-Hand Side of Chart);
```

Because Date/Time Stamps are expressed in seconds from midnight on January 1, 1970, the calculation results in seconds of time displayed between the left and right sides of the chart.

Whenever adding or subtracting from .ChartLength, time is expressed in seconds. Therefore, to subtract two hours from the current .ChartLength, convert hours to seconds before performing the calculation. For example: **(2 hours) \* (60 minutes/hour) \* (60 seconds/minute) = 7200 seconds.**

**Data Type**

Integer (read/write).

**Valid Values**

Any positive integer.

**Examples**

This example forces the length of the historical trend to one hour.

```
HtTag.ChartLength=3600 {60 minutes * 60 seconds/minute};
```

This example scrolls the trend left by 50 percent.

```
HtTag.ChartStart=HtTag.ChartStart - HtTag.ChartLength / 2;
```

This example scrolls the chart left by 10 percent.

```
HtTag.ChartStart=HtTag.ChartStart - (.10 * HtTag.ChartLength);
```

**See Also**

.ChartStart

## .ChartStart Dotfield

The .ChartStart dotfield can be used to set or verify the value of the starting (left side) date/time stamp of a historical trend.

**Category**

Historical.

**Usage**

```
HistTrendTag.ChartStart
```

**Parameter**

*HistTrendTag*

HistTrend tag assigned the name of the trend.

**Remarks**

This read/write dotfield is used to set or verify the value of the starting date/time stamp of a historical trend. The **.ChartStart** dotfield is expressed as the number of elapsed seconds after midnight January 1, 1970. The starting point is defined as the first date/time stamp on a historical trend.

**Data Type**

Integer (read/write).

**Valid Values**

Any positive integer.

**Example**

The following statement scrolls the chart to the right by one minute.

```
HtTagname.ChartStart=HtTagname.ChartStart + 60;
```

**See Also**

.ChartLength

**.Pen1-8 Dotfields**

The **.Pen1-8** dotfields assign a logged tag to a historical trend pen.

**Category**

Historical

**Usage**

```
HistTrendTag{.Pen1 | .Pen2 | .Pen3 | .Pen4 | .Pen5 | .Pen6 | .Pen7 | .Pen8};
```

**Parameter**

*HistTrendTag*

HistTrend tag assigned the name of the trend.

**Remarks**

You assign tags to trend pens using the **.Pen1-8** dotfields with the following format:

```
HistTrend.PenX = Tag_Name.TagID
```

Where X is an integer 1 to 8.

It is recommended that you use the HTSetPenName() and HTGetPenName() functions if possible.

---

**Note:** Only local tags can be assigned to a **.PenX** dotfield. The provider.tag notation cannot be used. The provider.tag can be used only with the HTSetPenName() function.

A good reference to use when learning how these dotfields work is a historical trend wizard placed on the screen and broken apart.

---

## Data Type

TagID (read/write).

## Valid Values

This dotfield data type is type **TagID**. This means only the handle of a tag can be assigned to the **.Pen1-8** dotfields. You cannot directly assign the name of a tag to the **.Pen1-8** dotfields. You must associate the associated **.TagID** dotfield of a tag to a **.Pen1-8** dotfield using the following syntax:

```
HistTrendTag.Pen1=LoggedTag.TagID;
```

In general, a TagID type tag can be equated only to another TagID tag. It cannot be used with any other tag type unless the **.TagID** dotfield extension is added to the other tag.

Although the **.Pen1-8** dotfields are considered read/write, their values cannot be directly shown on the screen.

## Examples

The following example assigns a new tag to the **.Pen5** dotfield of the historical trend associated with the Hist Trend tag. You must append the **.TagID** dotfield to the name of the logged tag in order to assign it to **.Pen5** dotfield.

```
HistTrendTag.Pen5=PumpPress.TagID;
```

Working from the previous example, you can show the name of the tag assigned to HistTrendTag.Pen5. Creating a legend that shows the tag assigned to each trend pen is useful information for an operator.

You cannot show the tag assigned to HistTrendTag.Pen5 in a Message Display link. The actual value of the **.Pen5** dotfield is an integer that represents a memory location within WindowViewer, which is not useful for display purposes. You need to create a new TagID type tag called **Pen05**. Place the following statement underneath the statement from the previous example:

```
Pen05=HistTrendTag.Pen5;
```

In the first example, the PumpPress tag is assigned to pen 5 of HistTrendTag. In this example, Pen05 is assigned the value of Pen5 of the HistTrendTag, which is the TagID of the PumpPress tag.

The **.Pen1-8** dotfields are pointers to the tags that are associated with pens selected to appear in a trend. The **.Pen1-8** dotfields are of a special data type, namely **.TagID**. After you make the assignment, you can use the **.Name** dotfield of the TagID tag to show the name of the tag.

## .TagID Dotfield

The **.TagID** dotfield can be used in conjunction with the **.Pen1 - .Pen8** dotfields to assign a tag to a historical trend pen.

### Category

Historical tag.

### Usage

```
tag_name.TagID
```

### Parameter

*tag\_name*

Any discrete, integer, real, indirect discrete, or indirect analog tag.

### Remarks

The **.TagID** dotfield provides the handle of a tag and is used mainly to assign tags to pens in a historical trend.

### Data Type

TagID (read-only).

### Example

This example uses the **.TagID** dotfield to assign the **PumpRPM** tag to pen 6 of the historical trend.

```
HistTrendTag.Pen6=PumpRPM.TagID;
```

### See Also

.Pen1-.Pen8

## .ScooterLockLeft Dotfield

The **.ScooterLockLeft** dotfield specifies whether an operator can move the right scooter further left than the left scooter's current position on the historical trend.

### Category

Historical.

### Usage

```
HistTrendTag.ScooterLockLeft
```

### Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.

### Remarks

In general, you should prevent an operator from moving the right scooter further left than the left scooter's current position. When the left scooter is locked, it forces the right scooter position to be equal to the left scooter position whenever the right scooter overtakes the left scooter.

### Data Type

Discrete (read/write).

### Valid Values

0 = False. Right scooter can move further left than the left scooter's current position on the historical trend

1 = True. Right scooter cannot move further left than the left scooter's current position on the historical trend.

### Example

The following example prevents the right scooter from moving further left than the left scooter's current position on the historical trend.

```
HistTrendTag.ScooterLockLeft=1;
```

### See Also

.ScooterPosRight, .ScooterPosLeft, .ScooterLockRight

## .ScooterLockRight Dotfield

The **.ScooterLockRight** dotfield specifies whether an operator can move the left scooter further right than the right scooter's current position on the historical trend.

### Category

Historical.

### Usage

```
HistTrendTag.ScooterLockRight
```

### Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.

### Remarks

In general, you should prevent the operator from moving the left scooter further right than the right scooter's current position. When the right scooter is locked, it forces the left scooter position to be equal to the right scooter position whenever the left scooter overtakes the right scooter.

### Data Type

Discrete (read/write).

### Valid Values

0 = False. Left scooter can move further right than the right scooter's current position on the historical trend.

1 = True. Left scooter cannot move further right than the right scooter's current position on the historical trend.

### Example

The following example prevents the left scooter from moving further right than the right scooter's current position on the historical trend.

```
HistTrendTag.ScooterLockRight=1;
```

### See Also

.ScooterPosRight, .ScooterPosLeft, .ScooterLockLeft

## .ScooterPosLeft Dotfield

The **.ScooterPosLeft** dotfield dynamically controls the position of the left scooter on a historical trend.

### Category

Historical

### Usage

```
HistTrendTag.ScooterPosLeft
```

### Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.



## Remarks

This read/write dotfield dynamically controls the position of the left scooter. You can use this dotfield in a QuickScript function to retrieve the current position of the left scooter, or you can assign a value to this dotfield to adjust the position of the left scooter to another location on the trend.

This dotfield is most often used in conjunction with the set of **HTGetValue()** functions. These functions must specify which historical trend is being queried, as well as the current position of the trend's scooters.

## Data Type

Real (read/write).

## Valid Values

0.0 to 1.0; where 0.0 is the extreme left of the historical trend chart, and 1.0 is the extreme right of the historical trend chart.

## Examples

The following example repositions the left scooter. The left scooter moves to a location 34 percent of the chart's total length from the left side of the historical trend chart currently associated with the MyHistTrendTag tag.

```
MyHistTrendTag.ScooterPosLeft=.34;
```

In the following statement, the QuickScript **HTGetValueAtScooter()** function retrieves the value of pen 1 at the left scooter's current position. A change to any value within a function's argument list causes the function to be re-evaluated. Each time the position of the left scooter changes, this statement is re-evaluated.

```
MyRealTag=HTGetValueAtScooter (MyHistTrendTag,MyHistTrendTag.UpdateCount, 1, MyHistTrendTag.ScooterPosLeft, 1, "PenValue");
```

## See Also

.ScooterPosRight, .ScooterLockLeft, .ScooterLockRight

## .ScooterPosRight Dotfield

The read/write **.ScooterPosRight** dotfield dynamically controls the position of the right scooter.

## Category

Historical.

## Usage

```
HistTrendTag.ScooterPosRight
```

## Parameter

*HistTrendTag*

HistTrend tag assigned the name of the trend.

## Remarks

This read/write dotfield dynamically controls the position of the right scooter. You can use this dotfield in a QuickScript function to retrieve the current position of the right scooter. You can also assign a value to this dotfield to move the right scooter to another location on the trend.

This dotfield is most often used in conjunction with the **HTGetValue()** functions. These functions must specify which historical trend is being queried, as well as the current position of the trend's scooters.

**Data Type**

Real (read/write)

**Valid Values**

0.0 to 1.0; where 0.0 is the extreme left of the historical trend chart, and 1.0 is at the extreme right the historical trend chart.

**Examples**

The following statement specifies a new location for the right scooter. The right scooter moves to a location 34 percent of the chart's total length from the left side of the Historical Trend chart associated with the MyHistTrendTag tag.

```
MyHistTrendTag.ScooterPosRight=.34;
```

The following statement uses the **HTGetValueAtScooter()** QuickScript function to retrieve the value of pen 1 at the right scooter's new position. A change to any variable within a function's parameter list causes the function to be re-evaluated. Each time the position of the right scooter changes, this statement is re-evaluated.

```
MyRealTag=HTGetValueAtScooter(MyHistTrendTag, MyHistTrendTag.UpdateCount, 2, MyHistTrendTag.ScooterPosRight, 1, "PenValue");
```

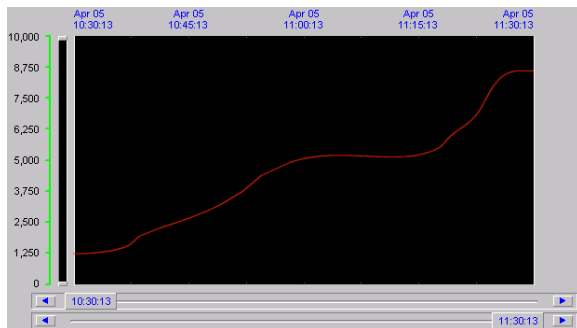
**See Also**

.ScooterPosLeft, .ScooterLockLeft, .ScooterLockRight

## Using the Historical Trend Wizard

The Historical Trend Wizard automatically creates a historical trend. Other than manually assigning tags to historical trend pens, the wizard automatically configures the historical trend using standard values.

The figure below shows the standard trend created with the Historical Trend Wizard. The trend includes sliders called scooters to show data at a specific location on the trend plot or zoom in on a selected range of trend data.



To add zoom and movement functions or pen controls to a historical trend, use the trend Zoom/Pan Panel and Trend Pen Legend wizards.

You can create and configure a historical trend. You can:

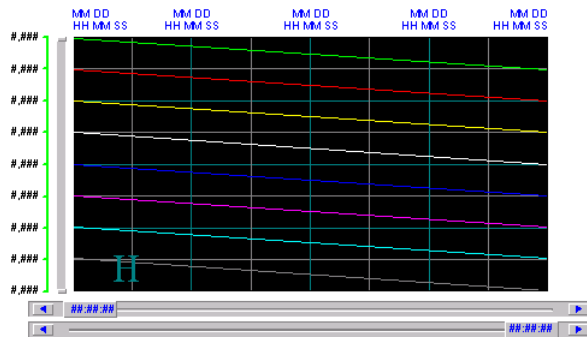
- Create a historical trend with wizards
- Select the tags for a trend
- Configure the historical trend time span
- Controlling a trend with QuickScripts

## Creating a Trend With the Historical Trend Wizard

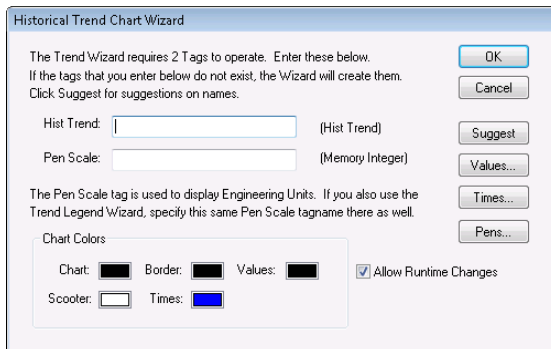
You can create a standard historical trend using the automation features of the Historical Trend Wizard. Other sections describe how to manually configure a historical trend by using wizard options.

### To create a historical trend with wizards

1. Open a window from WindowMaker to place a historical trend.
2. On the WindowMaker menu bar, click the **Wizard** icon. The **Wizard Selection** dialog box appears.
3. Select **Trends** from the list of wizards. The right pane of the **Wizard Selection** dialog box shows a set of trend wizard icons.
4. Select the **Hist Trend with Scooters** wizard and click **OK**. The **Wizard Selection** dialog box closes and your window reappears.
5. Move the cursor to the window location where you want to place the upper left corner of the historical trend. Click to place the trend in the window.



6. Double-click the trend. The **Historical Trend Chart Wizard** dialog box appears.



7. Click **Suggest**. The Historical Trend Chart Wizard automatically assigns default configuration values to the trend.

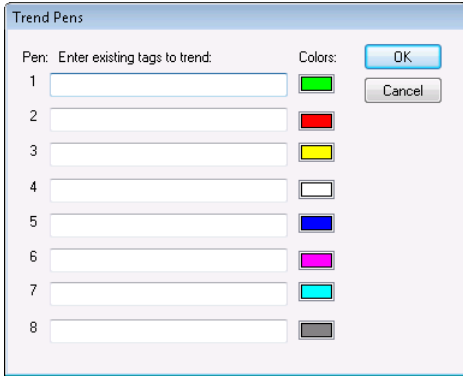
The only remaining configuration task is to assign tags to the trend pens.

## Configuring Which Tags to Display on the Trend Graph

Assigning tags to trend pens in the Historical Trend Chart Wizard is similar to the procedure for the Historical and Real-Time tools.

**To assign tags from the Historical Trend Chart Wizard**

1. If needed, double-click the historical trend. The **Historical Trend Chart Wizard** dialog box appears.
2. Click **Pens**. The **Trend Pens** dialog box appears.



3. Enter the name of an existing local tag in the **Pen** box. You can enter a maximum of 49 characters.

---

**Note:** WindowViewer must be closed. Otherwise, the Pen boxes cannot be selected.

---

If you double-click within the **Pen** box, the **Select Tag** dialog box appears with a list of tags assigned the **Log Data** option for the application. You can assign a tag to the pen by selecting it from the **Select Tag** dialog box.

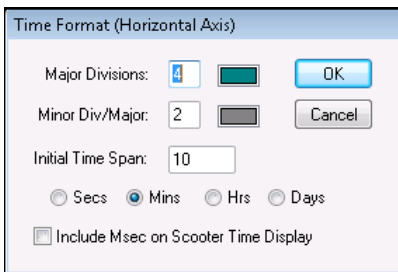
4. Click the color box next to each pen and select another color if you want to change the default pen color. Otherwise, skip this step and accept the default color.
5. Click OK to close the **Trend Pens** dialog box.
6. Click **OK** to close the **Historical Trend Chart Wizard** dialog box.

## Configuring the Historical Trend Time Span

The **Historical Trend Chart Wizard** dialog box includes an option to manually configure the time span shown from a trend created with the Historical Trend Wizard. You can manually configure a trend’s time instead of accepting the default configuration of the Historical Trend Wizard.

**To configure the time span of a historical trend**

1. Double-click the historical trend. The **Historical Trend Chart Wizard** dialog box appears.
2. Click **Times**. The **Time Format** dialog box appears.



3. Configure the time format. Do the following:

- a. In the **Major Divisions box**, type the number of major time divisions shown on the horizontal time axis of the trend.
- b. In the **Minor Div/Major box**, type the number of minor time divisions within each major division.
- c. In the **Initial Time Span box**, type the length of the time period shown on the horizontal axis of the trend. Trends created with the Historical Trend Wizard can be updated while the application is running in WindowViewer. Operators can change the length of the trend time period. But a historical trend always starts with the time period set from the **Time Format dialog box**.
- d. Select the unit of measure for the trend time period in seconds, minutes, hours, and days.
- e. Optionally, include milliseconds in the scooter time display. The following example shows a scooter slider with milliseconds appended to the current time.

11:30:13.000

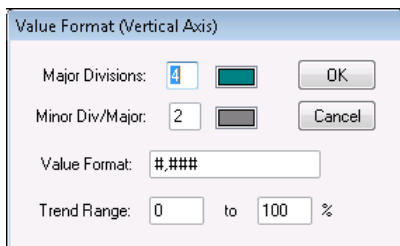
4. Click **OK** to close the **Time Format** dialog box.
5. Click **OK** to close the **Historical Trend Chart Wizard** dialog box.

## Configuring Display Options

The **Historical Trend Chart Wizard** dialog box includes an option to configure the vertical units of a historical trend. You can manually configure the major and minor value divisions shown on the vertical axis of a trend.

### To configure display options with the Historical Trend Chart Wizard

1. If necessary, double-click on the historical trend. The **Historical Trend Chart Wizard** dialog box appears.
2. Click **Values**. The **Value Format** dialog box appears with options to configure the vertical value axis of the trend.



3. Configure the value format. Do the following:
  - a. In the **Major Divisions box**, type the number of major value divisions shown on the trend’s vertical axis. Click the color box to access the color palette, and then click the color that you assign to the major value axis division lines.
  - b. In the **Minor Div/Major box**, type the number of minor divisions that you want to be visible within each major value axis division. Click the color box to access the color palette, and then click the color that you assign to the minor value axis division lines.
  - c. In the **Value Format box**, type the format of numbers that appear in the trend’s vertical value axis. The default number format is #,###.
  - d. In the **Trend Range boxes**, type the lower and upper percentage boundaries of a tag’s engineering units that appear in the trend.

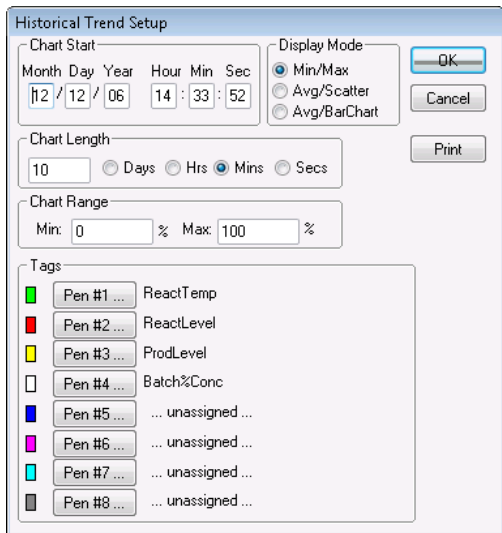
4. Click **OK** to close the **Value Format** dialog box.
5. Click **OK** to close the **Historical Trend Chart Wizard** dialog box.

## Changing the Configuration at Run Time

If you select the **Allow runtime changes** option when you configure your historical trend, operators can change some aspects of a historical trend during run time.

### To configure a historical trend during run time

1. Click the trend in WindowViewer. The **Historical Trend Setup** dialog box appears



2. In the **Chart Start** area, type the starting date and time of the chart.
3. In the **Display Mode** area, select the type of historical trend.
4. In the **Chart Length** area, type the length of time to show on the trend, and then select the time increment for the length.
5. In the **Chart Range** area, type the percentage of engineering unit scale shown as the vertical range of the trend.
6. In the **Tags** area, click each **Pen#** to assign a tag to the trend pen. The **Select Tag** dialog box appears and shows those tags for which logging is enabled.
7. Double-click on the name of a tag to assign it to the trend pen.
8. Click **OK** to save your run-time changes to the trend.

## Controlling a Historical Trend Wizard Using Scripts

You can use QuickScript functions with trend objects or animation link expressions to control a historical trend during run time. For example, you can use QuickScripts to update a trend to the current time, reassign tags to trend pens, connect pens to the chart, replot the grid, and remove or replot the scooters.

## Updating the Trend to the Current Time

You can create a script to update a historical trend to show recent tag data.

### HTUpdateToCurrentTime() Function

The **HTUpdateToCurrentTime()** function retrieves and shows the data with an end time equal to the current time. The start time is equal to end time minus the width of the chart.

#### Category

Historical

#### Syntax

```
HTUpdateToCurrentTime(Hist_Tag);
```

#### Argument

*Hist\_Tag*

HistTrend tag assigned the name of the historical trend.

#### Example

The following statement retrieves and shows data for the **Trend1** historical tag at the current time:

```
HTUpdateToCurrentTime("Trend1");
```

If the current time is 3:04 PM and the width of the trend is 60 seconds, the new end time is 3:04 PM. The new trend start time is 3:03 PM.

## Changing the Trend Configuration

You can use these script functions to change the tags assigned to the pens of a historical trend:

- *HTSelectTag()* Function on page 175
- *HTSetPenName()* Function on page 176

### HTSelectTag() Function

The **HTSelectTag()** function opens the **Select Tag** dialog box for the operator to assign a different tag to a trend pen.

---

**Note:** The **Select Tag** dialog box only lists the tags that are defined for historical logging with the **Log Data** option selected in the Tagname Dictionary.

---

#### Category

Historical

#### Syntax

```
HTSelectTag();
```

## Remarks

The **HTSelectTag()** function only shows tags in which the **Log Data** option has been selected from the Tagname Dictionary. However, it is possible to use the Tag Browser's filter to display a smaller set of tags. For example all tags that begin with "A". The function returns the selected tag and can be used as function parameter to assign a tag to a pen.

## Example

The following QuickScript causes the **Select Tag** dialog box to appear in WindowViewer. The user can then select a tag from the list. This tag is assigned to pen 1 by the Historical Object named HistTrend.

```
HTSetPenName("HistTrend",1,HTSelectTag());
```

## See Also

### HTSetPenName()

## HTSetPenName() Function

The **HTSetPenName()** function assigns a different tag to a trend's pen.

## Category

Historical

## Syntax

```
HTSetPenName(Hist_Tag,PenNum,Tagname);
```

## Arguments

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*PenNum*

Integer tag or value representing the pen number (1-8) of the trend.

*Tagname*

Name of the new tag to assign to the pen.

## Remarks

This QuickScript function is the only method to add tags from a distributed history provider during run time.

You can enter a maximum of 49 characters for a reference in a pen name.

You may see the following error message when you attempt to unassign a trend pen:

```
VIEW /UpdateData: Invalid DBS.TAGNAME handle - 0
```

This error occurs if you're trying to unassign a pen that was previously assigned to a remote tag in the form *histprovider.tag\_name*. To resolve this error, create a local tag with the **Log Data** option selected. Then, use the following script to unassign the pen:

```
HTSetPenName("HistTrend", 1, "localtag" );
{assigns the pen to a locally logged tag---localtag}
HistTrend.Pen1=None;
{unassigns the pen}
```

Where None is a TagID type tag.



## Examples

The following statement assigns the OutletPressure tag to pen 3 of Trend1.

```
HTSetPenName("Trend1", 3, "OutletPressure");
```

The following statement assigns the **HistPrv1.Tag1** tag to TrendPen4 of Trend1.

```
HTSetPenName("Trend1", TrendPen4, "HistPrv1.Tag1");
```

## See Also

**HTSelectTag()**

## Retrieving Information About the Trend and Historical Data

You can create scripts that retrieve information from a historical trend while it is running. Use the following functions:

- *HTGetPenName()* Function on page 177
- *HTGetTimeAtScooter()* Function on page 178
- *HTGetTimeStringAtScooter()* Function on page 178
- *HTGetValue()* Function on page 179
- *HTGetValueAtScooter()* Function on page 180
- *HTGetValueAtZone()* Function on page 181
- *HTScrollLeft()* Function on page 182
- *HTScrollRight()* Function on page 182
- *HTZoomIn()* Function on page 183
- *HTZoomOut()* Function

## HTGetPenName() Function

The **HTGetPenName()** function returns the name of the tag currently assigned to the specified pen number of the historical trend.

### Category

Historical

### Syntax

```
MessageResult=HTGetPenName(Hist_Tag, UpdateCount, PenNum);
```

### Arguments

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield. The argument value acts as a data change trigger to re-evaluate the function

*PenNum*

Integer tag or value representing the pen number (1-8) of the trend.

### Example

The following statement retrieves the name of the tag assigned to Pen 2 of the Trend1 trend and places the name in the **TrendPen** message tag:

```
TrendPen=HTGetPenName("Trend1", Trend1.UpdateCount, 2);
```

## HTGetTimeAtScooter() Function

The **HTGetTimeAtScooter()** returns the time in seconds after 00:00:00 hours GMT, January 1, 1970 for the sample at the scooter location specified by the *ScootNum* and *ScootLoc* arguments.

### Category

Historical

### Syntax

```
IntegerResult=HTGetTimeAtScooter(Hist_Tag, UpdateCount, ScootNum, ScootLoc);
```

### Arguments

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield.

*ScootNum*

Integer representing the left or right scooter:

1=Left Scooter

2=Right Scooter

*ScootLoc*

Real number representing the value at the **.ScooterPosRight** or **.ScooterPosLeft** positions on the trend.

### Remarks

Any changes to the values assigned to the UpdateCount, ScootNum, and ScootLoc arguments cause the expression to be evaluated. This ensures the expression is evaluated after new data retrievals or after a scooter is moved.

### Example

The following statement retrieves the time in seconds for the value at the current left scooter location of the Trend1 trend:

```
HTGetTimeAtScooter("Trend1", Trend1.UpdateCount, 1, Trend1.ScooterPosLeft);
```

## HTGetTimeStringAtScooter() Function

The **HTGetTimeStringAtScooter()** function returns the string containing the time/date for the sample at the specified scooter location.

### Category

Historical

**Syntax**

```
MessageResult=HTGetTimeStringAtScooter(Hist_Tag, UpdateCount, ScootNum, ScootLoc,  
Format_Text);
```

**Arguments***Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield.

*ScootNum*

Integer representing the left or right scooter:

1=Left Scooter

2=Right Scooter

*ScootLoc*

Real number representing the value at the **.ScooterPosRight** or **.ScooterPosLeft** positions on the trend.

*Format\_Text*

String specifying the time/date format to use. The following **Format\_Text** strings are acceptable:

"Date", "Time", "DateTime", "DOWShort" (Wed, for example), and "DOWLong" (Wednesday, for example).

**Remarks**

Any changes to the values assigned to the **UpdateCount**, **ScootNum**, and **ScootLoc** arguments cause the expression to be evaluated. This ensures the expression is evaluated after new data retrievals or after a scooter is moved. The format of the string determines the contents of the return value.

**Example**

The following statement retrieves the date and time for the value at the current scooter location for the right scooter of the Trend1 trend. The value is stored in the **NewRightTimeString** message tag and is in "Time" format:

```
NewRightTimeString=HTGetTimeStringAtScooter ("Trend1", Trend1.UpdateCount, 2,  
Trend1.ScooterPosRight, "Time");
```

## HTGetValue() Function

The **HTGetValue()** function returns a value of the requested type for the trend's specified pen.

**Category**

Historical

**Syntax**

```
RealResult=HTGetValue(Hist_Tag, UpdateCount, PenNum, ValType_Text);
```

**Arguments***Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield.

*PenNum*

Integer tag or value representing the pen number (1-8) of the trend.

*ValType\_Text*

String indicating the type of value to return:

PenAverageValue = Average for the entire trend.

PenMaxValue = Maximum pen value for the entire trend.

PenMinValue = Minimum pen value for the entire trend.

PenMaxEU = Maximum engineering units value for the entire trend.

PenMinEU = Minimum engineering units value for the entire trend.

PenStdDev = Standard deviation for the entire trend.

**Remarks**

The function returns the requested value as a real value.

**Example**

The following statement obtains the standard deviation for the pen 2 data retrieved from the PumpPress trend. The value is stored in the LeftHemisphereSD memory real tag:

```
LeftHemisphereSD=HTGetValue("PumpPress", PumpPress.UpdateCount,2,"PenStdDev");
```

**HTGetValueAtScooter() Function**

The HTGetValueAtScooter() function returns a value of the requested type for the sample at the specified scooter position, trend, and pen number. The UpdateCount argument causes the expression to be evaluated after function processing is finished.

**Category**

Historical

**Syntax**

```
RealResult=HTGetValueAtScooter(Hist_Tag, UpdateCount,ScootNum,ScootLoc,PenNum,ValType_Text);
```

**Arguments**

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield.

*ScootNum*

Integer representing the left or right scooter:

1 = Left Scooter

2 = Right Scooter

*ScootLoc*

Real number representing the trend's **.ScooterPosRight** or **.ScooterPosLeft** dotfields.

*PenNum*

Integer tag or value representing the pen number (1-8).

*ValType\_Text*

String indicating the type of value to return:

PenValue = Value at scooter position.

PenValid = 0 if value is invalid, 1 if valid.

When the *ValType\_Text* argument is used with the `HTGetValueAtScooter()` function, use one of the valid types listed.

**Example**

The following function returns a 1 if the value is an actual sample or a 0 if the value is invalid for pen 3 of the Trend1 trend for the right scooter's current position:

```
HTGetValueAtScooter("Trend1",Trend1.UpdateCount, 2,Trend1.ScooterPosRight,3, "PenValid");
```

**HTGetValueAtZone() Function**

The `HTGetValueAtZone()` function returns a value of the requested type for the data located between the right and left scooter positions for a trend's specified pen.

**Category**

Historical

**Syntax**

```
RealResult=HTGetValueAtZone(Hist_Tag,UpdateCount, Scoot1Loc,Scoot2Loc,PenNum,ValType_Text);
```

**Arguments**

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield. It is used only as a trigger to evaluate the function.

*Scoot1Loc*

Real representing the trend's **.ScooterPosLeft** dotfield. It is used only as a trigger to evaluate the function.

*Scoot2Loc*

Real representing the trend's **.ScooterPosRight** dotfield. It is used only as a trigger to evaluate the function.

*PenNum*

Integer tag or value representing the pen number (1-8) of the trend.

*ValType\_Text*

String indicating the type of value to return.

PenAverageValue = Average for zone between the scooters.

PenMaxValue = Maximum value for the zone between the scooters.

PenMinValue = Minimum value for the zone between the scooters.

PenMaxEU = Maximum engineering unit value for the zone between scooters.

PenMinEU = Minimum engineering unit value for the zone between the scooters.

PenStdDev = Standard Deviation for the zone between the scooters.

## Remarks

A real value is returned representing the calculated value of the given type. Specifying constant values for the Scoot1Loc and Scoot2Loc arguments has no effect and are only used to trigger the evaluation of the function. The function uses the trend tag's .ScooterPosLeft and .ScooterPosRight dotfield values directly, regardless of the values you specify for the Scoot1Loc and Scoot2Loc arguments.

## Example

The following statement calculates the average value for data between the right and left scooters of the Trend1 trend for pen 1. The value is stored in the AvgValue memory real tag:

```
AvgValue=HTGetValueAtZone("Trend1", Trend1.UpdateCount,Trend1.ScooterPosLeft,
Trend1.ScooterPosRight,1, "PenAverageValue");
```

## Panning and Zooming the Trend

You can create QuickScripts containing functions that select specific data from a historical trend during run time.

### HTScrollLeft() Function

The **HTScrollLeft()** function sets the start time of the trend to an earlier time than the current start time by a percentage of the trend's total time span. The effect is to scroll the chart to the left to an earlier time by a specified percentage of the trend's total time span.

#### Category

Historical

#### Syntax

```
HTScrollLeft(Hist_Tag,Percent);
```

#### Arguments

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*Percent*

Real number representing the percentage of the chart's time span to scroll (0.0 to 100.0) left.

#### Example

The following statement scrolls the time/date left by 10 percent of the PumpPress trend's total width:

```
HTScrollLeft("PumpPress",10.0);
```

If the current display starts at 12:00:00 PM and the display width is 60 seconds, then the new trend starts at 11:59:54 AM after the function is processed.

### HTScrollRight() Function

The **HTScrollRight()** function sets the start time of the trend to a time later than the current start time by a percentage of the trend's width. The effect is to scroll the date/time of chart to the right by a specified percentage of the trend's width.

#### Category

Historical

**Syntax**

```
HTScrollRight(Hist_Tag,Percent);
```

**Arguments**

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*Percent*

Real number representing the percentage of the chart to scroll (0.0 to 100.0) right.

**Example**

The following statement scrolls the PumpPress trend to the right by 20 percent.

```
HTScrollRight("PumpPress", 20.0);
```

If the current display starts at 12:00:00 PM and the display width is 60 seconds, then the new trend starts at 12:00:12 PM after the function is processed.

**HTZoomIn() Function**

The HTZoomIn() function calculates a new chart width and start time. If the trend's scooters are at the left and right sides of the trend, then the new chart width equals the old chart width divided by two. The new start time is calculated based on the value of the *LockString* argument.

If the scooters are not at the left and right sides of the trend, the HTZoomIn() function zooms the trend to the zone defined by the scooters and ignores the *LockString* argument.

**Category**

Historical

**Syntax**

```
HTZoomIn(Hist_Tag,LockString);
```

**Arguments**

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*LockString*

String representing the type of zoom:

- StartTime            Keep the start time equal to before zoom
- Center              Keep center time equal to before zoom
- EndTime             Keep end time equal to before zoom

**Remarks**

If the scooter positions are not at the left and right sides of the trend, the new chart width is the time between .ScooterPosLeft and .ScooterPosRight positions. In this case, the value of LockString is not used. The minimum chart width is one second. The scooter positions are set to .ScooterPosLeft=0.0 and .ScooterPosRight=1.0 after the zoom.

**Example**

The following statement zooms the display by a factor of two and maintains the same start time for the Trend1 trend. Trend1.ScooterPosRight is equal to 1.0 and Trend1.ScooterPosLeft is equal to 0.0. If the start time before zooming was 1:25:00 PM and the chart width was 30 seconds, the new start time remains at 1:25:00. The new chart width is 15 seconds.

```
HTZoomIn("Trend1", "StartTime");
```

**HTZoomOut() Function**

The **HTZoomOut()** function calculates a new chart width and start time. The new chart width is the old chart width multiplied by two. The new start time is calculated based on the value of the *LockString* argument .

**Category**

Historical

**Syntax**

```
HTZoomOut(Hist_Tag, LockString);
```

**Arguments***Hist\_Tag*

HistTrend tag assigned the name of the trend.

*LockString*

String representing the type of zoom:

StartTime = Keep start time equal to before zoom

Center = Keep center time equal to before zoom

EndTime = Keep end time equal to before zoom

**Remarks**

The current scooter positions have no effect on HTZoomOut(). After the function zoom finishes, the new scooter positions are set to .ScooterPosLeft=0.0 and .ScooterPosRight=1.0.

**Example**

The following statement zooms out the trend time by a factor of two and maintains the same center time for the Volume trend. If the start time before zooming was 2:15:00 PM and the chart width was 30 seconds, the start time after zooming is now 2:14:45. The chart width is 60 seconds and the center of the trend remains at 2:15:15.

```
HTZoomOut("Volume", "Center");
```

**Printing the Trend**

You can print a historical trend currently visible from a WindowViewer screen using the PrintHT() function in a script.



## PrintHT() Function

The **PrintHT()** function prints the historical trend currently visible on the screen. Usually, the **PrintHT()** function is associated with a screen button included on the historical trend window. Operators click the button to print the visible historical trend with its current values.

### Category.

Historical

### Syntax

```
PrintHT(Trend_Tag);
```

### Argument

*Trend\_Tag*

HistTrend tag.

### Example

This example prints the PumpPress historical trend currently visible on the screen.

```
PrintHT(PumpPress);
```

## Troubleshooting the Trend

You can create QuickScripts to verify if data was successfully retrieved that appears in a historical trend. Use the **HTGetLastError()** function to troubleshoot the trend.

## HTGetLastError() Function

The **HTGetLastError()** function can be used in a script to determine if an error occurred during the last data retrieval for a specified historical trend pen.

### Category

Historical

### Syntax

```
[Result=]HTGetLastError(Hist_Tag,UpdateCount, PenNum);
```

### Arguments

*Hist\_Tag*

HistTrend tag assigned the name of the trend.

*UpdateCount*

Integer representing the trend's **.UpdateCount** dotfield.

*PenNum*

Integer tag or value representing the pen number (1-8) of the trend.

*Result*

Integer assigned to a tag that represents the status of the last script function call for the specified pen.

0 = No error

1 = General server error

- 2 = Old request
- 3 = File error
- 4 = Server not loaded
- 5 = Trend/Pen passed in function does not exist
- 6 = Trend tag does not exist in database
- 7 = Pen number passed to function is invalid (not in range of 1 to 8).
- 8 = No tag or a non-logged tag assigned to the pen number

### Examples

The following statement retrieves the status of the last data retrieval for pen 3 of the Trend1 trend and assigns the result to the ResultCode integer tag.

```
[ResultCode=]HTGetLastError("Trend1", Trend1.UpdateCount, 3);
```

In an animation Analog Value Display QuickScript the following statement would be used:

```
HTGetLastError("Trend1", Trend1.UpdateCount, 3);
```

## Displaying Real-Time Values in a Trend

You can create real-time trends by two methods. The Real-Time Trend object provides a standard set of controls to select the data, set a time range, and specify the physical appearance of the graph. InTouch also includes the 16-Pen Trend Wizard, which is an optional control to create real-time and historical trends. For more information about creating real-time trends with the 16-Pen Trend Wizard, see *Creating a 16-Pen Trend* in the InTouch® HMI Supplementary Components Guide.

## Using Real-Time Trend Objects

You can create a real-time trend to show current values in your application.

### Creating a Real-Time Trend

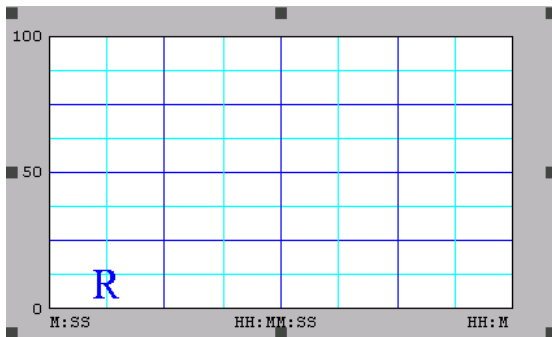
You can use the Real-time Trend tool to create a trend object in a window. The first time you paste a real-time trend object, WindowMaker uses default settings. After configuring a real-time trend, WindowMaker uses the last configuration values as the initial settings for any new real-time trend object.

You can draw the trend chart any size within the borders of the window.

#### To create a real-time trend

1. Select the **Real-time Trend** tool from the Drawing Toolbar.

2. Move the mouse over the window area where you want to place the real-time trend. Drag the mouse diagonally to create a rectangle the desired size of the trend. The Real-time Trend object appears in the window.



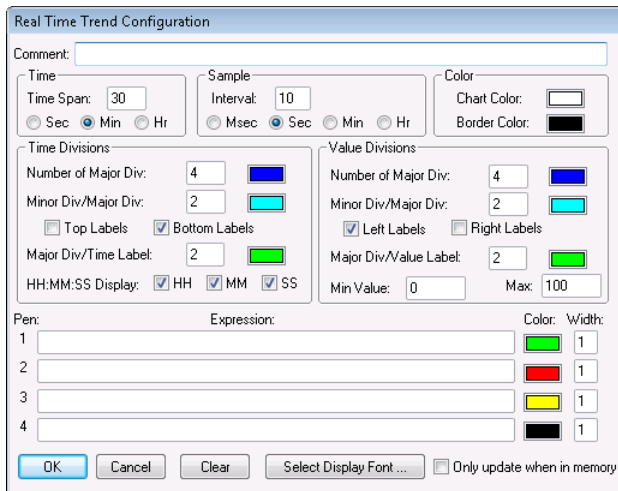
3. If needed, adjust the height and width of the trend with its object handles.

## Configuring Which Tags to Display on a Real-time Trend

A real-time trend pen creates a graphical representation of current data from any local tag or an expression that contains one or more local tags. You configure the pens that show tag data in a real-time trend.

### To configure real-time trend tags

1. Double-click the trend object in the window. The **Real Time Trend Configuration** dialog box appears.



2. In the **Expression** area, type the name of a local tag or expression that contains one or more local tags.  
If you double-click in the **Pen** box, the **Select Tag** dialog box appears showing a list of tags defined for the application. You can assign a tag to the pen by selecting it from the **Select Tag** dialog box.
3. Click the color box next to each pen assigned a tag to show a color palette.
4. Click the color that you want to assign to the pen.
5. In the **Width** box, type the line width in pixels for each pen shown in the trend.  
Selecting a line width greater than 1 increases the time required to update or print a trend.

6. Select the **Only update when in memory** check box if you want the trend to update only when shown in the active window.

If you do not select this option, the trend updates continuously even if the window is closed. Continuous trend updating slows system performance.

7. Keep the **Real Time Trend Configuration** dialog box open and go to the next procedure described in *Configuring the Real-Time Trend Time Span and Update Rate* on page 188.

## Configuring the Real-Time Trend Time Span and Update Rate

You can configure the time span and update rate of a real-time trend.

### To set a real-time trend time span and update rate

1. If needed, double-click the trend object. The **Real Time Trend Configuration** dialog box appears.
2. In the **Time** area, type the length of time in the **Time Span** box that you want to appear on the horizontal x-axis of the trend.
3. Select the unit of measure for trend time.
  - Seconds (Sec)
  - Minutes (Min)
  - Hours (Hr)

For example, if you enter 30 in the **Time Span** box and then select **Min**, the time span shown on the chart is 30 minutes.

4. In the **Sample** area, type a number in the **Interval** box that the trend expression is evaluated and the chart updates.
5. Select the interval unit of measure.
  - Milliseconds (Msec)
  - Seconds (Sec)
  - Minutes (Min)
  - Hours (Hr)

For example, if you enter 10 in **Interval** and then select **Sec**, the real-time trend is updated every 10 seconds.

6. Keep the **Real Time Trend Configuration** dialog box open and go to the next procedure described in *Configuring Real-time Trend Display Options* on page 188.

## Configuring Real-time Trend Display Options

You can configure the visual appearance of a real-time trend.

### To configure real-time trend display options

1. Double-click on the trend object. The **Real Time Trend Configuration** dialog box appears.
2. In the **Color** area, configure the color. Do any of the following:
  - Click the **Chart Color** box to open the color palette. Select the background color for the trend.

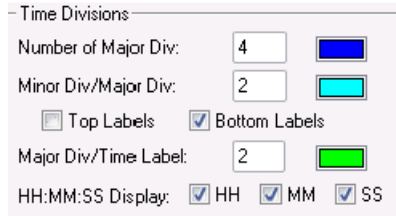
White is the default background color. Any other background color significantly increases the time needed to print a trend.

- Click on the **Border Color** box to open the color palette. Select the border color of the trend.

3. In the **Time Divisions** area, configure the time divisions. Do the following:

- In the **Number of Major Div box**, type the number of major trend time divisions. The major time divisions appear on the horizontal time axis of the trend.

The number of major time divisions must be an even multiple of the **Major Div/Time Label** value. For example, a division number of 20 is an even multiple of the **Major Div/Time Label** value of 4.



- Select the color for the major division lines.
- In the **Minor Div/Major Div box**, type the number of minor time divisions visible within each major time division.

The number of minor time divisions should be evenly divisible within the major division period. For example, if the major division period is set to 60 seconds, entering a value of 2 in **Minor Div/Major Div** creates two minor time division periods of 30 seconds.

- Select the color for the minor division lines.
- Select either the **Top Labels** or **Bottom Labels** check box to specify the placement of time labels on the trend.

You can select both options to place time labels at the top and bottom of the trend. Leaving both options blank removes time labels from the horizontal axis of the trend.

- If you are using time labels, type the number of major time division lines per time label in **Major Div/Time Label**. The number of major divisions must be an even multiple of the **Major Div/Time Label** value.

Select the color of the time division labels.

- Select the time units shown as part of the major time division label.

- Hours (HH)
- Minutes (MM)
- Seconds (SS)

4. In the **Value Divisions** area, configure the appearance of the vertical axis of the trend.

Value Divisions

Number of Major Div: 4 [Color: Blue]

Minor Div/Major Div: 2 [Color: Cyan]

Left Labels  Right Labels

Major Div/Value Label: 2 [Color: Green]

Min Value: 0 Max: 100

**Value Divisions** options are configured like **Time Divisions** options. The major and minor divisions on the y-axis show the magnitude of data values rather than time. The vertical axis specifies the range of data values that appear in the trend based upon engineering units for all tags.

To show decimal points for the minor and major value divisions, type real numbers for the **Min Value** and **Max** options. For example, 0.00 to 100.00.

5. Click **Select Display Font**. The **Font** dialog box appears with options to set the font, style, and size of text that appears in the trend.
6. Click **OK**.

## Printing a Trend at Run Time

Several factors determine how fast a trend can be printed. The primary factor is the size of the trend on the printed page. The display mode of the trend also affects printing performance. Min/Max or Average/Scatter trends can be printed more quickly than Average/Bar Chart trends. Also, the longer and wider the lines on the trend are, the longer it takes to print.

Another factor that affects printing performance is the background color of the trend. In most cases, a white background prints more quickly than a colored background.

## Configuring Trend Printing Options

You can configure options that determine how a trend is printed.

**To configure historical trend printing**

1. On the **Special** menu, click **Configure** and then **Historical Logging**. The **Historical Logging Properties** dialog box appears.

Historical Logging Properties

2. In the **Printing Control** area, specify the percentage of the page to print the trend in the **Default % of page to print on** box.

If you enter 50, the trend is printed on half of the page vertically and horizontally. A trend printed at 50 percent takes much less time to print than a full-page trend.

As a printing alternative, you can use the **PrintWindow()** QuickScript function.

3. In the **Max consecutive time to spend printing** box, type the process time slice in milliseconds.

A time slice represents the period allocated to the computer processor to run the print module process in the foreground and print the trend. A longer time slice enables the trend to be printed more quickly at the expense of other processes running on the computer.

4. In the **Time to wait between printing** box, type the time in milliseconds the print module waits between processor time slices.  
A shorter waiting period between processor time slices enables the trend to be printed more quickly.
5. Click **Select Printer Font**. The **Font** dialog box appears. Select the characteristics of the text appearing in a trend.
6. Click **OK** to save your printing configuration and close the **Historical Logging Properties** dialog box.

## Displaying Historical Tag Values from Other InTouch Nodes or IndustrialSQL Server

If you want to use data stored remotely to create historical trends, the remote provider must be registered in the InTouch history provider list. This list specifies the name and network location of each history provider. These names are referred to whenever historical trend pens point to tags at the remote history provider.

You can define a remote history provider and assign historical trend pens to tag data stored at the remote location. You can:

- Configure remote history providers.
- Configure pens to display data from a remote history provider.
- Assign pens to tag data stored at a remote history provider using the Tag Browser.
- Assign a pen to a remote tag using a QuickScript.

For more information about using data from a remote history provider, see *Distributing Applications* in the InTouch® HMI Application Management and Extension Guide.

## Using the InTouch HMI with the IndustrialSQL Server

The IndustrialSQL Server is a real-time, relational database designed specifically for industrial applications. You can optionally configure the Historical Logger to store InTouch historical data to a IndustrialSQL Server database.

---

**Note:** For more information about logging InTouch historical data to a database, see the IndustrialSQL Server documentation. For more information about setting up the InTouch HMI with a remote history provider, see *Distributing Applications* in the InTouch® HMI Application Management and Extension Guide.

---

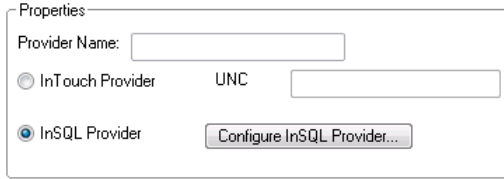
If you use the IndustrialSQL Server to store historical data, you must use the Distributed Name Manager from WindowMaker to specify a connection to the database.

To configure a connection to a **IndustrialSQL Server** database

1. Open WindowMaker.
2. On the Tools view, expand the **Configure list**.
3. **Select Distributed Name Manager**. The **Distributed Name Manager** dialog box appears.
4. On the **Distributed History** tab, type **InSQL** as the new provider in the **Provider Name** box.



5. Select **InSQL Provider**.

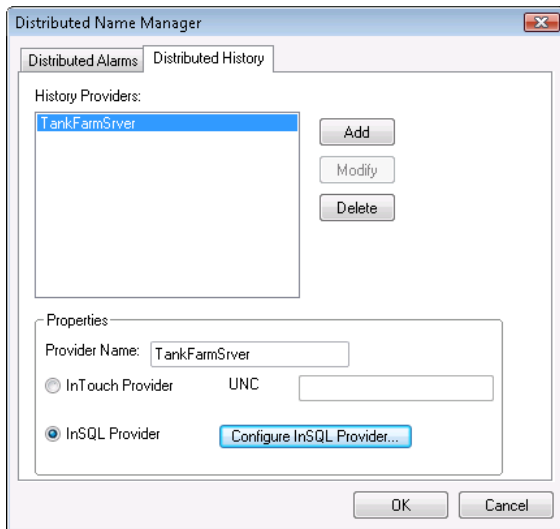


6. Click **Configure InSQL Provider**. The **InSQL History Provider Properties** dialog box appears.

- a. In the **Data Source** box, type the node name of the server where the IndustrialSQL Server is installed.
- b. Enter a name for a IndustrialSQL Server database user account.
- c. Enter the password for the user account in both the **Password** and **Re-enter password** boxes.
- d. Click **Test** to verify the connection to the IndustrialSQL Server database. A message appears indicating whether the connection to the database is successful or not.

7. Click **OK** to close the dialog box.

8. Click **OK** to close the **InSQL History Providers** dialog box. The IndustrialSQL Server node appears in the **History Providers** list.



9. Click **OK** to close the **Distributed Name Manager** dialog box.

## Configuring Pens to Display Remote Trend Data

Historical trends can display tag data from both local and remote history providers. You can assign trend pens to display data from a remote history provider.

### To display a tag from a remote history provider

1. Double-click on the historical trend to show the **Historical Trend Configuration** dialog box.
2. In each pen's **Tagname** box, type the reference to a remote history provider. The format of the reference to a remote history provider is:

*history\_provider\_name.tag\_name*

**Example:**

```
TankFarm1.Pump1RPM
```

Each pen of a historical trend can refer to a different remote history provider.

3. Click **OK** to save your configuration changes.

---

**Note:** The .TagID dotfield cannot be used in remote history provider tag references.

---

## Using the Tag Browser to Assign Pens to Remote History Providers

The following procedure explains how to use the Tag Browser to assign a trend pen to tag data from a Remote History provider. Using the Tag Browser to select tags eliminates the need to manually enter each tag name and reduces the likelihood of errors.

The remote node name you specify in the Access Name does not have to be the actual name of the node where the tag resides. But, you must create an Access Name to define the remote history provider as a tag source. For more information about creating an Access Name, see *Setting Up Access Names* on page 61.

### To define a remote history provider as a tag source

1. Create an Access Name that specifies the node name where the history provider is located.
2. Double-click the historical trend to open the **Historical Trend Configuration** dialog box.
3. Double-click a pen's **Tagname** input box to show the **Select Tag** dialog box.
4. Click **Define Tag Sources** to define the remote history provider as a tag source.
5. Click the **Tag Source** arrow and select the new remote history provider tag source in the list, or click the **Tree View** button and select the tag source in the tree view pane. The **Select Tag** dialog box refreshes and shows the tags from the selected remote history provider.
6. Select the tag that you want to assign to the historical pen and click **OK**. The **Historical Trend Configuration** dialog box reappears with the selected tag listed in the pen's **Tagname** box as: *AccessName: Item*.
7. Replace the AccessName: portion with the history provider name you defined in the **Distributed Name Manager**.

For example, *HistPrv1.tag\_name*

This process may seem cumbersome, but after you have defined the history provider as a tag source in the Tag Browser, each time you double-click another tag input box, you simply double-click the tag name in the **Tag Browser**, and then replace the AccessName: portion with the history provider name.

In WindowViewer, if run-time changes are allowed for the historical trend, when the user clicks a pen button to change the tagname, the Tag Browser appears but only the local application's tags are accessible.

## Using a QuickScript to Assign a Pen to a Remote History Provider

While an InTouch application is running, you can configure a trend pen to show tag data from a remote history provider. Create a QuickScript that specifies the remote history provider tag reference in the `HTSetPenName()` function. For example:

```
HTSetPenName("HistTrendTag", 1, "HistPrv1.Boiler1");
```

In this example, the number 1 specifies the number of the pen in the historical trend that plots the remote **Boiler1** tag values from the HistPrv1 remote history provider.

The run-time **Historical Trend Setup** dialog box and **Pen** dotfields are not supported for remote history providers.

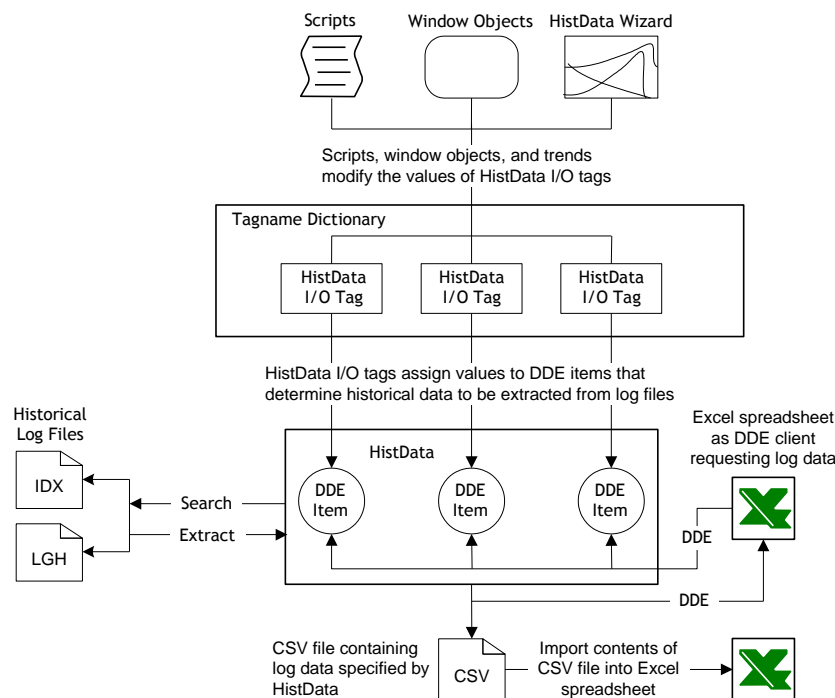
## Chapter 11

# Accessing Historical Tag Values from Other Applications

## About Accessing Historical Tag Values from Other InTouch Applications

You can use the InTouch HistData utility to extract data from historical log files to a comma separated value (.csv) file. Applications like Excel can extract InTouch log data directly from HistData as a DDE client or import log data from the output file created by the HistData utility.

The figure below shows the process to save selected historical log data to a file or a DDE client application.



## Using DDE Items to Show Historical Data

The HistData program includes a set of DDE items that specify how historical data is extracted from log files. These items are part of the HistData internal database. You assign a value to each item.

The following table summarizes HistData items defined in the HistData program.

Item	Data Type	Descriptions
<b>DATADIR</b>	Message	Path of the folder containing historical log files.
<b>DBDIR</b>	Message	Path of the folder containing the contents of the InTouch Tagname Dictionary.
<b>STARTDATE</b>	Message	Start date to extract data from the log file. The format of the start date is MM/DD/YY.
<b>STARTTIME</b>	Message	Start time to extract data from log files. The format of the start time is HH:MM:SS using a 24-hour clock.
<b>DURATION</b>	Message	<p>Length of the data collection interval from log files. <b>DURATION</b> can be expressed as:</p> <ul style="list-style-type: none"> <li>• Weeks (w)</li> <li>• Days (d)</li> <li>• Hours (h)</li> <li>• Minutes (m)</li> <li>• Seconds (s)</li> </ul> <p>Fractional <b>DURATION</b> periods can be specified. For example, <b>DURATION</b>=0.5m is equivalent to 30 seconds. To request a single sample, set <b>DURATION</b> to 0.</p>
<b>INTERVAL</b>	Message	<p>Length of time between data collection intervals. <b>INTERVAL</b> can be expressed in weeks, days, hours, minutes, and seconds. The units of time of an <b>INTERVAL</b> period are the same as a <b>DURATION</b> period.</p> <p>Fractional intervals can be specified. For example, <b>INTERVAL</b>=0.25d represents 6 hours.</p> <p>The maximum period for <b>DURATION</b> or <b>INTERVAL</b> is six weeks. The maximum six week period applies to any time value assigned to <b>DURATION</b> or <b>INTERVAL</b>. For example, 42 is the maximum number of days of an <b>DURATION</b> or <b>INTERVAL</b> period.</p>
<b>FILENAME</b>	Message	Name and folder location of the file containing data extracted from the historical log file.

Item	Data Type	Descriptions
<b>WRITEFILE</b>	Integer	Flag that indicates the status of HistData write operation to the output file. When set to 1, HistData writes the requested data to the file specified by the <b>FILENAME</b> Item Name. When the file update is complete, <b>WRITEFILE</b> automatically resets to 0.
<b>ERROR</b>	Message	String containing a description of the last error that occurred while extracting data from log files. When <b>STATUS</b> is set to 1, the <b>ERROR</b> string is set to None. When <b>STATUS</b> is set to 0, the <b>ERROR</b> string contains an error message.
<b>TAGS1, TAGS2,...</b>	Message	<p>String containing the name of one or more tags whose data is extracted from the log files.</p> <p>The <b>TAGS</b> string can be 131 characters in WindowViewer and 255 characters in Excel.</p> <p>The string can be appended for longer requests by adding tag items named Tagsn, where <i>n</i> represents an incrementing integer.</p> <p>If a tag needs additional tag text, place a plus (+) at the end of the string.</p> <p>For example:</p> <pre>TAGS="\$Date,ProdLevel,ProdTemp,+" TAGS1="ReactLevel,Temp,GasLevel,+" TAGS2="MotorStatus"</pre> <p>Duplicate tags are not allowed and the maximum length of each tag's string is 512 bytes.</p>
<b>PRINTTAGNAMES</b>	Discrete	Flag that indicates whether the names of tags are placed above the associated column of values. When set to 1, tag names are printed. When set to 0, tag names are not printed.
<b>DATA</b>	Message	This item holds the requested data in the HistData program in comma separated values format. It is used by other applications to <b>ADVISE</b> or <b>REQUEST</b> data by DDE.
<b>STATUS</b>	Discrete	<p>Status of the most current HistData operation.</p> <p>A value of 1 indicates HistData successfully extracted historical data from the log file. A value of 0 indicates that an error occurred.</p>

Item	Data Type	Descriptions
<b>SENDDATA</b>	Integer	<p>Flag that indicates the status of the HistData update operation. When set to 1, HistData updates the <b>DATA</b> item with the requested data. When the update is complete, <b>SENDDATA</b> automatically resets to 0.</p> <p>If you receive an error message indicating too much data was requested using <b>SENDDATA</b>, shorten the <b>DURATION</b> period or reduce the number of requested tags. Duplicate tags are not allowed and the maximum length of each tag's string is 512 bytes.</p>

## Accessing Log Data with DDE

You can use two methods to extract log data to an output file.

- Use the manual method if you want to save historical log data from eight or more tags to the output file.
- Use the HistData Wizard instead if you only want to save log data mapped to the pens currently assigned to a historical trend.

## Manually Extracting Log Data with HistData

You can manually extract log data to an output file. Complete the steps in the following order.

- Create a HistData Access Name
- Create I/O tags for HistData
- Create a HistData window
- Run HistData

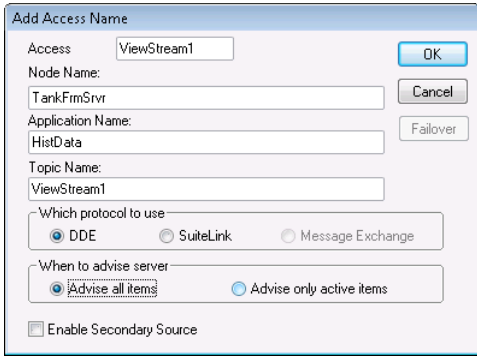
### Create a HistData Access Name

For InTouch to request data from the HistData program, you must define an Access Name.

#### To define an access name

1. On the **Special** menu, click **Access Names**. The **Access Names** dialog box appears.

2. Click **Add**. The **Add Access Name** dialog box appears.



3. In the **Access Name** box, type a name up to 32 alphanumeric characters. The values assigned to **Access Name** and **Topic Name** should be the same.
4. In the **Node Name** box, type the name of the node where the log files are currently located.
5. In the Application Name box, type HistData without the .exe file name extension.
6. In the Topic Name box, type the name you specified from the **Access Name** box. **Access Name** and **Topic Name** should be the same.
7. Select **DDE** as the communication protocol.
8. In the **When to advise server** area, select **Advise all items** whenever HistData is used.
9. Click **OK**.

## Create HistData Tags

After defining an Access Name, create the following I/O type tags to generate one output file that contains the data from the log files. Assign the Access Name created in the previous step to the tags.

Tag	I/O Tag Type	Item
<b>HDWDATADIR</b>	Message	DataDir
<b>HDWDBDIR</b>	Message	DbDir
<b>HDWDURATION</b>	Message	Duration
<b>HDWERROR</b>	Message	Error
<b>HDWFILENAME</b>	Message	FileName
<b>HDWINTERVAL</b>	Message	Interval
<b>HDWSTARTDATE</b>	Message	StartDate
<b>HDWSTARTTIME</b>	Message	StartTime
<b>HDWSTATUS</b>	Message	Status
<b>HDWTAGS,</b> <b>HDWTAGS1,</b>	Message	Tags



Tag	I/O Tag Type	Item
<b>HDWTAGS2</b>		
<b>PRINTTAGNAMES</b>	Discrete	PrintTagNames
<b>HDWWRITEFILE</b>	Integer	WriteFile

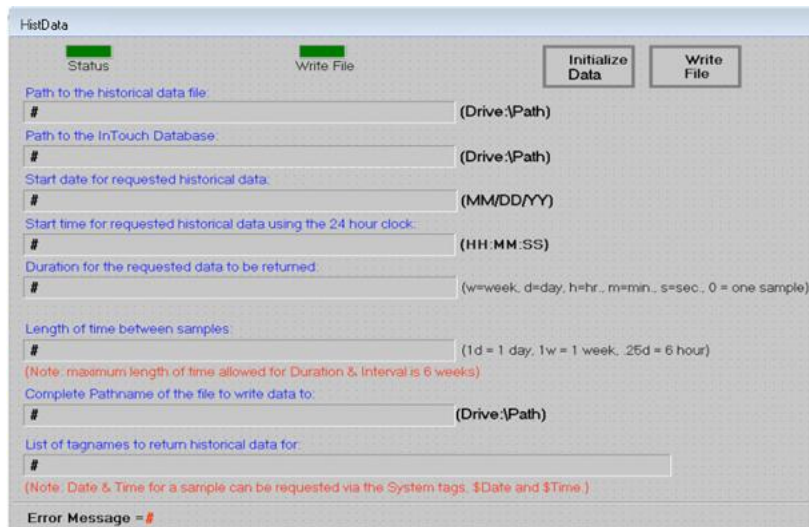
**Note:** The HistData Wizard creates these tags automatically except for the PRINTTAGNAMES tag.

Create two additional tags if you want to send log data to the Data item so that it can be accessed from other applications. Also, the HistData Wizard does not automatically create the HDWSendData and HDWData tags.

Tag	I/O Tag Type	Item
<b>HDWSendData</b>	Discrete	<b>SendData</b>
<b>HDWData</b>	Message	<b>Data</b>

## Create a HistData Window

After you create the I/O type tags, create a new window called **HistData** similar to the following example:



The # symbols are linked to a user input link. For example, the # symbol has a User Inputs/String link to the HDWDataDir tag. The user input link allows you to change the value of the tags during run time.

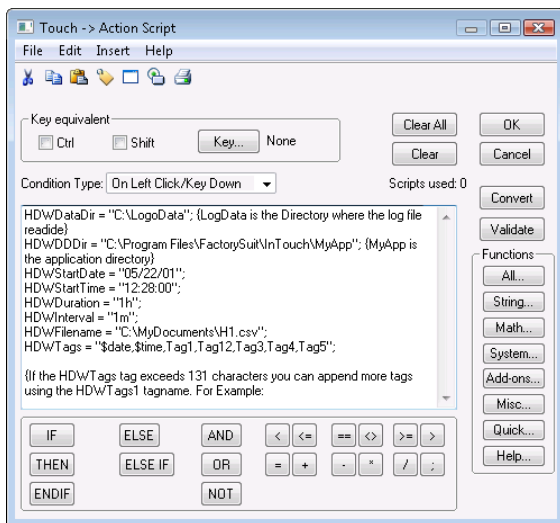
The **Status** button is linked to a fill color—discrete expression, based on the HDWStatus tag.



The **Write File** button is linked to a fill color—discrete expression, based on the HDWWriteFile tag.

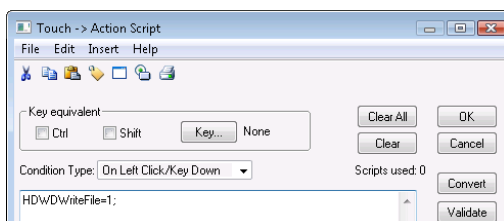


The **Initialize Data** button is linked to a Touch Pushbutton Action script.



When the **Initialize Data** button is clicked, the HistData items are initialized with the desired values. If needed, these values can also be changed during run time by using the User Inputs Links.

The **Write File** button is linked to a Touch Pushbutton Action script:



When clicked, the **WriteFile** button generates the output file.

## Run HistData

After creating the HistData window, complete the following steps to run it under WindowViewer.

### To run the HistData window

1. Start HistData and minimize it.
2. Start WindowViewer and open the HistData window.
3. Click the **Initialize** button and make changes to the HistData items if needed.
4. Click the **WriteFile** button.

If the operation is successful, the value of Status is ON and the color associated with the ON status appears. If the operation is not successful, the value of Status is OFF and Error Message shows the cause of the failure.

## Using the HistData Wizard to Extract Log Data

You can create an output file containing log data that appears in a historical trend. InTouch includes the HistData Wizard to automate the steps to extract data from a log file.

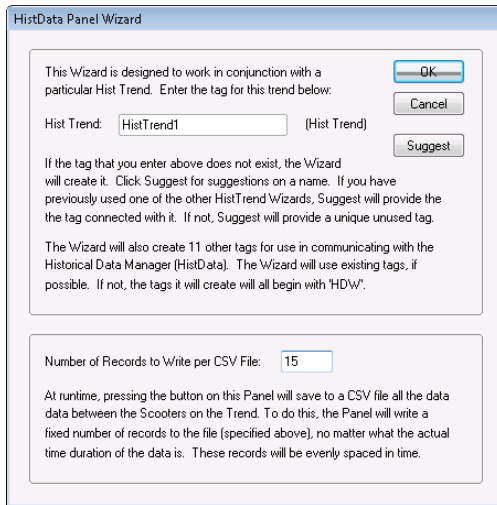
Because the HistData output file writes log data shown in a historical trend, the file can only contain data from the tags currently assigned to the pens of the historical trend.

### To use the HistData Wizard to extract log data

1. Start WindowMaker.
2. Open a window that includes a historical trend.
3. Click the **Wizards** tool from the menu bar. The **Wizard Selection** dialog box appears.
4. Select the **Trends** group from the left pane.
5. Select the **HistData Wizard** icon from the right pane and click **OK**.
6. Move your mouse pointer over an area of the trend window where you want to place the HistData object.
7. Click to place the HistData object in the trend window. The HistData Wizard creates a window object consisting of a button and the **Filename** box that shows the path where the output file will be created.



8. Double-click the HistData Wizard object placed in the trend window. The **HistData Panel Wizard** dialog box appears.



9. In the **Hist Trend** box, type a name for the HistTrend tag.
10. In the **Number of Records to Write per CSV File** box, type the number of records to write to an output file.
11. Click **OK**. The HistData Wizard creates a set of tags that are identified with a HDW prefix.  
The HistData Wizard creates the tags listed in *Create HistData Tags* on page 200. The HistData Wizard assigns the tags to the HisDataViewSt Access Name.
12. Run the historical trend window with WindowViewer.
13. Click **Save to File** that is part of the HistData window object. HistData creates the output file in the folder location shown in the window object.

---

**Note:** HistData does not populate the .csv file properly if the tag values do not change within an hour of starting InTouch with the newly-created \*.idx and \*.lgh files.

The HistData data does not have the same resolution as for historical trend scooters. The HistData resolution is based on the number of values requested within the time span. It is not an exact reflection of the values that are in the \*.idx and \*.lgh files.

---

## Accessing Historical Data from Other Applications

You can write Excel macros to extract data from a HistData file. The HistData program responds to the INITIATE, POKE, and TERMINATE functions within the macro. The POKE function with a keyword (an internal database item) sets the parameters that define a query. After the query is properly specified, run the macro to request the selected historical data from the HistData file.

The following example shows a macro written with VBA:

```

Sub GetHistdata()
    Dim rangeToPoke
    Dim channelNumber

    channelNumber = Application.DDEInitiate("histdata", "topic")
    Set rangeToPoke = Worksheets("Sheet1").Cells(1, 1)
    Application.DDEPoke channelNumber, "DataDir", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(2, 1)
    Application.DDEPoke channelNumber, "DEDir", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(3, 1)
    Application.DDEPoke channelNumber, "StartDate", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(4, 1)
    Application.DDEPoke channelNumber, "StartTime", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(5, 1)
    Application.DDEPoke channelNumber, "Duration", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(6, 1)
    Application.DDEPoke channelNumber, "Interval", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(7, 1)
    Application.DDEPoke channelNumber, "FileName", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(8, 1)
    Application.DDEPoke channelNumber, "Tags", rangeToPoke
    Set rangeToPoke = Worksheets("Sheet1").Cells(9, 1)
    Application.DDEPoke channelNumber, "WriteFile", rangeToPoke
    Application.DDETerminate channelNumber
End Sub
    
```

In the example above, the data to be poked is in Sheet1. The following example shows the data to be poked:

	A	B	C	D	E	F	G	H	I
1	"C:\Program Files\FactorySuite\InTouch\histtest"								
2	"C:\Program Files\FactorySuite\InTouch\histtest"								
3	"07/22/07"								
4	"01:00:00"								
5	"1h"								
6	"1m"								
7	"C:\Program Files\FactorySuite\InTouch\F10.csv"								
8	\$DATE,\$TIME,\$tag1,\$tag2,\$tag3"								
9	1								
10									
11									
12									
13									

## Troubleshooting HistData Errors

You may see errors that can occur when extracting log data with HistData. The following table lists typical HistData problems or error messages in the left column. The right column describes possible causes and solutions to the problem.

Error Messages or Condition	Cause and/or Solution
<p><b>Error Message:</b> Too much data requested – shorten the duration or reduce the number of tagnames.</p>	<p>This error occurs <b>when too</b> much data is requested by the <b>SendData</b> item. If the only purpose is to create an output file containing data from the log files, do not use the <b>SendData</b> item.</p>
<p><b>Error Message:</b> Could not open file C:\FILES1\HISTDATA.CSV</p>	<p>The folder path does not exist or the spelling of the folder path is incorrect.</p>
<p><b>Error Message:</b> Could not open file C:\FILES\</p>	<p>No output file is defined.</p>
<p><b>Error Message:</b> DATADIR item invalid</p>	<p>The destination folder path specified by the <b>DataDir</b> item does not exist. Verify the spelling of the folder path.</p>
<p><b>Error Message:</b> STARTDATE item invalid</p>	<p>The <b>StartDate</b> item contains an invalid format for the starting date. From Windows, change the computer’s date format to mm/dd/yy.</p>
<p><b>Error Message:</b> No log files found</p>	<p>There are no log files for the requested date in the path specified by the <b>DataDir</b> item.</p>
<p><b>Error Message:</b> Could not find tagname TAG• in database</p>	<p>The requested tag does not exist in the application’s Tagname Dictionary. Verify the name of the tag is spelled correctly.</p>
<p><b>Error Message:</b> Could not find tagname.x in: C:\IT6.0B\HISTEST</p>	<p>The file <b>tagname.x</b> does not exist or it is corrupted.</p>
<p>No output .csv file is created and no errors appear.</p>	<ul style="list-style-type: none"> <li>• HistData is not running.</li> <li>• The <b>Tags</b> item does not list any tags that have been specified for logging.</li> <li>• The HDWWritefile is incorrectly defined. Make sure the tag is an integer tag, the DDE Access Name is correct, and the item is <b>WriteFile</b>. Also, make sure there is no scaling where <b>MinEU=MinRaw</b> and <b>MaxEU=MaxRaw</b>.</li> </ul>

Error Messages or Condition	Cause and/or Solution
<p>The output .csv file contains date and time stamps but does not contain any logged data for the requested tags.</p>	<p>There are no entries in the Historical log for the tags during the requested time period. Display a historical trend to verify if the log file contains data during the requested period.</p>
<p>The WWLogger contains the following message:            Error for DDE            HistData Viewstream1!            WriteFile: Poke was rejected by the server.</p>	<p>The error message is written to the WWLogger each time the creation of the .csv file fails because of errors assigning values to the HistData items. This error can also occur if you try to set the <b>WriteFile</b> item to 0, or if you try to write to the error item.</p>
<p>The .csv file contains only a single record when there should be many records from the log file.</p>	<p>The <b>Interval</b> item may be assigned an incorrect value, which creates a very small collection interval.            Also, the <b>Duration</b> item may have an invalid format such as HDWDuration=1- (no increment specified).</p>

## Appendix A

# IEEE Decimal Units

## About IEEE Decimal Units

The Historian HMI uses the Institute of Electrical and Electronics Engineers (IEEE) 754 standard to transform 32-bit binary values to floating-point decimal numbers.

IEEE 754 32-bit numbers are stored in 16-bit Programming Logic Controllers (PLCs) as two 16-bit words. The floating point registers in the PLC are usually sequential in their numbering scheme for the low and high hexadecimal words. The current generation of 32-bit personal computers use a single 32-bit register. The register's bit numbering scheme follows the same format as two sequential 16-bit registers.

To use floating point numbers in an Historian application, the Historian must be able to transform the values stored in the two 16-bit PLC registers. Bit conversions must be made because Historian always regards the raw PLC register values as individual integers. It is not possible to perform a Boolean **AND** operation of the two register integers and transform them into a real number. Historian cannot perform a type conversion for dual integer registers.

## Showing Floating Point Numbers in the Historian HMI

The Historian HMI uses the IEEE 32-bit floating point format to show real numbers in an application. The IEEE floating point format is only an approximation of an actual real number. Unless the real number is an even power of two, it cannot be represented exactly using the IEEE 32-bit floating point format. The precision of an IEEE 32-bit floating point number is approximately eight decimal places.

When you want to show a real number in an Historian application, make sure the number does not exceed eight digits. The following floating-point number formats show valid real numbers within an Historian application:

- #
- #,###
- #.##
- 0#
- ###.##
- #.#####
- ###.#####
- #####.##



ABCDEF

###.####

Any floating-point numbers with more than eight digits are subject to rounding errors.

If you do not include a decimal in the format of the text, then the number is displayed with decimals, per the real format decimal precision configured in WindowViewer's Advanced Format properties.

---

**Note:** If you add "#" to the left of a decimal, or if there is no decimal, do not limit the number of digits displayed.

---

### Example 1

A Historian application should show the real number 2.3. But, the number 2.3 is not an even power of two and cannot be precisely represented by the IEEE 32-bit floating point format beyond 8 decimal digits.

To ensure the value 2.3 is shown from the application as the ASCII characters 2.3, the number must not exceed eight digits. If the number exceeds the eight digit maximum, the resulting number may be shown as 2.29999999 or 2.30000001 instead.

### Example 2

When two real tags values are compared, the difference of two real tag values should be greater than FLT\_EPSILON (value 1.19209290E-07F, in decimal 0.0000001192092896). However if the number exceeds 8 digits the resulting value may be incorrect. To correct for this multiply the value by 1000 or a larger multiple of 10. By doing this the value is be greater than 1e-7. Perform the necessary comparison operations and then divide by 1000, or the number you multiplied by.

## Appendix B

# InTouch Licensing

## About InTouch Licensing

InTouch licenses are based on varying numbers of tags you can use while running an application. You need to understand how tags are counted with the InTouch license scheme. You can use a set of functions to calculate the anticipated number of remote reference tags in your applications.

---

**Important:** Licensing numbers are subject to change at any time.

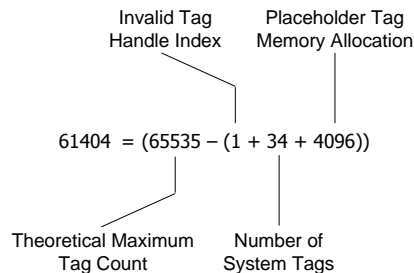
---

## Understanding License Tag Counts

While an InTouch application is running, tag handles are stored in a 64K memory database. Each tag must be assigned a handle. Tag handles are initialized and used by WindowViewer, but never saved permanently to disk after the application stops.

This run-time database can theoretically store 65535 tag handles, which includes both local tags and tags that reference a remote tag source. An InTouch application can never have more simultaneously active tags than the available memory handles in this run-time database. Your InTouch license determines how many local and remote tags can be assigned handles within the run-time database.

Also, the maximum number of active tags in an application is restricted by the functional limits of the run-time database. The actual maximum tag count is less than the theoretical maximum of 65535 tag handles. The following figure shows the actual InTouch maximum tag count. A set of constants is subtracted from the maximum potential tag count.



- The invalid tag handle bit is reserved to indicate if an invalid handle value occurs within the WindowViewer run-time database.
- InTouch version 10 includes 34 system tags, which cannot be replaced by user-defined tags. If you migrate a version 7.11 or earlier application to the current version of InTouch, the system tag count is 37.

- At configuration time, 4096 database handles are reserved to store placeholder tags. When you import windows, scripts, or symbols during configuration time, placeholder tags are assigned to this memory segment. During run time, all 4096 placeholder handles are available to be assigned to remote reference tags.

InTouch license options are based upon the maximum number of local and remote reference tags that can be used in an application.

If an InTouch tag license is for less than 60K tags, then a sticky tag licensing scheme is enforced. A sticky tag is a remote tag reference that is bound at run-time when WindowViewer receives a data change notification for the remote reference. WindowViewer updates remote tag references during run-time up to the maximum limit of the InTouch license. WindowViewer does not update any additional remote tag references beyond the license limit. WindowViewer does not decrement the remote reference tag count when a window is closed. Each remote reference tag count sticks while the application is running.

A single message appears when you exceed the maximum remote reference tag count of your InTouch license. After the license maximum is reached, the values associated with invalid remote reference tags are never updated in the application. You must stop and restart the application before you can open other windows that include one or more remote tag references that are not already associated with those counted against the license limit.

A license that permits 60K tags means that sticky tags are not enforced and there is no enforced limit to the number of tags or the combinations of local and remote tags that can be used within an InTouch application. The total maximum number of tags is not limited to 60,000. Instead, the maximum possible number of remote reference and local tags is based upon the implementation limit of the InTouch run-time database, which is slightly larger than 60K.

Using a 60K licence, the implementation limits for local and remote reference tags are:

- Total possible local tags  
 $61404 = 65535 - (4096 + 1 + 34)$
- Total possible remote reference tags  
Maximum =  $65535 - (1 + 34 + \# \text{ Local Tags})$

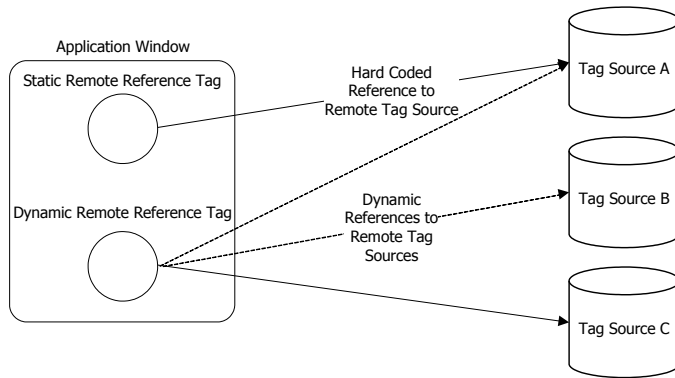
So, running an application with a 60K license, you are effectively trading a potential remote reference for every user-defined tag in the local tag database. You can never have less than 4096 possible remote reference tags available in a run-time configuration.

## Understanding InTouch Remote Reference Limits

There are two types of InTouch remote reference tags. A *static* remote reference is hard coded to a fixed remote address when you define the tag from the Tagname Dictionary. A static remote reference is assigned a tag handle in the tag database when the application starts running. A static remote reference tag count sticks while the application is running.

A *dynamic* remote reference resolves the target address while the application is running. If the dynamic remote reference tag is assigned a database handle, the target address can be changed during run time by using the .Reference dotfield or `IOSetRemoteReference()` function within a script.

The following figure shows an example of an InTouch application running under a 60K license without sticky remote reference tags. The count for the static remote reference tag sticks while the application is running. But, the count for the dynamic remote reference tag is only for the active tag source. The previous connections to remote tag sources do not stick and are not counted in the remote reference or total tag count.



The InTouch 60K license does not use sticky tag counts that impose limits on the number of dynamic remote tag references. This allows an application to dynamically access more than 60K tags during the period the application is running. The tag use count for dynamic remote references fluctuates up and down as windows with remote references are opened and closed. But, the application can never have more simultaneously active tags than the implementation limit of the run-time tag database.

Dynamic reference tag counts only fluctuate up and down when WindowViewer uses disk storage to save the contents of a running application. If WindowViewer is configured to cache InTouch windows or Industrial graphics, the remote reference tag counts may not decrease unless a window is removed from the cache.

When a window is not visible does not mean the remote tag references are not still bound to their sources. However, if window caching is completely disabled, and no high priority windows have been specified, then WindowViewer operates much like the legacy "Always load from disk" scenario. In this case, all windows are removed from memory when they are closed and the dynamic remote tag references are reclaimed in a 60K license environment.

A remote reference from an I/O tag is not included in the sticky remote reference count of the InTouch license. An I/O tag's remote reference can change an unlimited number of times without counting against the sticky remote reference limit.

When fail-over to the secondary tag source occurs, the application can access the same items from the secondary source without increasing the licensed tag count. After failover, accessing new items from the secondary tag source increases the tag count. These items are accessible after fail-back to the primary tag source.

After the tag count reaches the licensed maximum, no further items can be activated regardless of whether they are accessed from the primary or secondary tag source.

## Remote Tag Count Functions

The InTouch HMI includes a set of functions to verify if your applications conform to the remote tag requirements of your license. You can write temporary scripts that include these functions to test and identify any potential licensing issues with your application before deploying it into production. After you verify your application does not have any licensing issues, remove the scripts.

## IORRGetSystemInfo() Function

The IORRGetSystemInfo() function returns a tag count for a running InTouch application. Based upon an argument value, the IORRGetSystemInfo() function returns a numerical value, which can be:

- Maximum number of remote tag addresses specified by the InTouch license
- Number of remote tag addresses counted against the license over the period that an InTouch application has been running
- Number of remote tags currently activated in an InTouch application
- Number of available remote tags in a running InTouch application
- Number of remote reference tags, which are currently disabled
- Number of local tags in a running InTouch application
- -1 if an error occurs during the function call or the *Option* argument is assigned an invalid value.

### Category

Miscellaneous

### Syntax

```
IORRGetSystemInfo(Option);
```

### Argument

#### *Option*

An integer tag or integer constant that specifies the type of remote reference tag count to return. Possible values are:

1	Returns the maximum number of permitted remote tag addresses based upon the InTouch license. Local I/O tags are not counted in the remote tag count.  This number is constant while the InTouch application is running.
2	Returns the number of unique remote tag addresses counted against the licensed limits that are activated while an InTouch application is running. Local I/O tags are not counted in the remote tag count.  If the license permits more than 60000 remote reference tags, this number may be 0, regardless of how many remote tag addresses are activated. While running under an unlimited license, WindowViewer does not count the activated remote tag addresses.  While running an application under a license that has a remote tag limit, this count increments up to the limit of the remote tag license count. After the remote tag limit is reached, no further remote tag addresses can be activated. Only addresses currently activated can be reactivated. Use IORRWriteState with the <i>Option</i> argument set to 3 to obtain a list of remote reference tags that count against the license limit.
3	Returns the number of remote reference tags currently activated within an InTouch application.

4	Returns the number of remote reference tags that can be activated in an InTouch application without running out of remote tag handles. This count typically changes while an application is running. Stopping and starting scripts and opening and closing windows containing remote reference tags affect the remote reference tag count.  This number can be less than the number still remaining on the license, especially if the license is unlimited. This occurs because there is an internal limit to how many remote reference tags can be active simultaneously.
5	Returns the number of remote tags currently in the disabled state within an InTouch application.
6	Returns the number of local tags currently in the InTouch application.

**Example**

The following example returns the number of remote reference tags counted against the license limit while an InTouch application is running. The returned remote reference tag count is assigned as the value of the RRTagCount integer tag.

```
RRTagCount = IORRGetSystemInfo(2);
```

**IORRWriteState() Function**

The IORRWriteState() function saves information about the current state of an application’s remote tags to a text file. The function creates the file if it does not exist. Each time the script containing the function runs, new information is concatenated to the file.

You can specify what remote tag information is saved to the file. Also, the function’s return value indicates whether information is added successfully to the file.

**Category**

Miscellaneous

**Syntax**

```
IORRWriteState(FilePath, Option, " ");
```

**Arguments**

*FilePath*

Full folder path to the text file containing information about an application’s remote tags. The *FilePath* argument can be a string constant or a message tag.

*Option*

An integer tag or integer constant that specifies the remote tag count information written to the file. Possible values are:

1	List of current remote tag addresses. The information also includes each remote tag’s state, activation time, and deactivation time.
2	List of all currently active addresses, including activation time.

3	<p>List of all remote reference tag addresses that have been activated and counted against the license limit.</p> <p>New items are added to the list as new remote tag addresses are activated. When the license limit is reached, no further items are added to the list.</p> <p>However, no addresses are added to this list if the remote tag license limit is unlimited.</p>
4	<p>List of current addresses not activated because the remote reference tag count exceeds the license limit or because the internal tag handle limit was reached.</p> <p>Does not return addresses related to licensing if the remote tag license limit is unlimited.</p> <p>If the licensing is unlimited, the list contains any items that are not currently active because of implementation limitations. If any item in this list becomes active, it is removed from the list. When an item is deactivated because of licensing limitations, it is removed from the list. This list updates while the InTouch application is running.</p>

*Empty String " "*

This argument is reserved for future use, but must be included with the IORRWriteState() function in a script.

## Results

These are some examples to interpret information saved to the file by the IORRWriteState() function call.

### Current Addresses Listing

The following line from the output file shows an example of a fully activated remote reference tag, which can be updated. This line shows that address 65535 is assigned to the TestProt:di000 remote reference tag:

```
65535 <TestProt:di000> (RAA) {C:5/23/2007 9:58:35 AM} {A:5/23/2007 9:58:35 AM}
```

Following the remote reference tag name are three flags enclosed within parentheses:

- The first flag, R, indicates the tag’s remote reference has been successfully resolved. The first flag is assigned a value of X if the tag’s remote reference is still pending.
- The second flag indicates if the remote tag is currently active, A, or inactive, D.
- The third flag indicates if the address is allowed, A, or disallowed, D, because of licensing restrictions.

The date following C indicates when the remote tag was created. The date following A shows when the tag was most recently activated. The line includes a trailing deactivation time if the remote reference tag has been deactivated while the InTouch application is running.

The following line from the output file shows an example of an active remote reference tag that exceeds the tag count limit of the InTouch license. The tag’s remote reference is successfully resolved and the tag is currently active:

```
65414 <TestProt:di121> (RAD) {C:5/23/2007 9:58:35 AM} {A:5/23/2007 9:58:35 AM}
```

But, no tag value updates occur within the InTouch application because the address assigned to the remote reference tag exceeds the tag count limit of the InTouch license.

### Active Address Listing

The following line from the output file shows an example of a fully activated remote tag whose assigned values update the InTouch application:

```
65429 <TestProt:di106> (A) {C:5/23/2007 9:58:35 AM} {A:5/23/2007 9:58:35 AM}
```

The first number is the remote tag handle, followed by the address, then the flags, A, for allowed or D for disallowed. The creation, most recently activated, and most recently deactivated times follow the flags in the output line. The deactivation time does not appear if the remote tag has never been deactivated while the application is running.

The following line from the output file shows an example of an active remote tag that exceeds the license limits:

```
65342 <TestProt:di193> (D) {C:5/23/2007 9:58:35 AM} {A:5/23/2007 9:58:35 AM}
```

### Licensed Addresses

The following line from the list shows the address assigned to the remote reference tag and when it was added to the list.

```
<testprot:di000> {C:5/23/2007 9:58:36 AM}
```

### Denied Addresses

Denied addresses appear in the list because of implementation limitations, or because the tag count exceeds license maximum.

This example shows a remote tag address, which exceeds the license limit:

```
testprot:di125 [1] (L) {F:5/23/2007 9:58:39 AM} {R:5/23/2007 9:58:39 AM}
```

The address is listed along with the count to indicate how many times an attempt was made to reference the item. The flag indicates if the address is in the list because of an exceeded license, L, or because of an internal implementation limit, I. The two times represent the first time it was added to the list and its most recent access time.

### Example

This example writes the current activated remote tag addresses to a file located in the c:\intouch\data folder. The ReturnValue tag is assigned an integer, which indicates whether the function call successfully wrote remote tag information to the file.

```
ReturnValue = IORRWriteState("c:\intouch\data", 2, "");
```

## IORRGetItemActiveState() Function

The IORRGetItemActiveState() function returns the status of a specified remote tag address.

### Category

Miscellaneous

### Syntax

```
IORRGetItemActiveState(ItemPath, Option);
```

### Arguments

*ItemPath*

*ItemPath* is a string that represents the address of interest. *ItemPath* can be a string constant or a message tag.



*Option*

An integer tag or integer constant that specifies the type of remote reference tag count to return. Possible values are:

1	<p>Determine if a current remote tag address is currently active.</p> <p>The return value is 1 if the address is current and active. The return value is -1 if the address is not current.</p> <p>The return value is 0 if the address is current but inactive.</p>
2	<p>Determine if a current remote tag address has ever been activated while an application is running.</p> <p>The return value is 1 if the address is current and has been activated at least once. The return value is -1 if the address is not current. The return value is 0 if the address is current and never been activated.</p>
3	<p>Determine if a current remote tag address has ever been deactivated.</p> <p>The return value is -1 if the address is not current.</p> <p>The return value is 0 if the address is current and never been deactivated.</p>
4	<p>Determine if a current remote tag address is disabled.</p> <p>The return value is 1 if the address is current and has been deactivated at least once. The return value is -1 if the address is not current. The return value is 0 if the address is not disabled. The return value is 1 if the address is current and is disabled.</p>
5	<p>Determine if the address is in the allowed list.</p> <p>The return value is 0 if the address is not in the list.</p> <p>The return value is 1 if the address is in the list.</p>
6	<p>Determine if the address is in the disallowed list.</p> <p>The return value is 0 if the address is not in the list.</p> <p>The return value is 1 if the address is in the list.</p>

**Examples**

This example determines if the TestProt:di000 remote tag address is currently active:

```
ReturnValue = IORRGetItemActiveState("TestProt:di000", 1);
```

This example determines if the TestProt:di121 remote tag address is currently disabled:

```
ReturnValue = IORRGetItemActiveState("TestProt:di121", 4);
```

This example determines if the TestProt:di001 remote tag address is currently counted against the license limit.

```
ReturnValue = IORRGetItemActiveState("TestProt:di001", 5);
```

# Index

## \$

- \$AccessLevel system tag • 22, 32
- \$ApplicationChanged system tag • 32
- \$ApplicationVersion system tag • 32
- \$Date system tag • 32
- \$DateTime system tag • 32
- \$Day system tag • 32
- \$HistoricalLogging system tag • 32
- \$Hour system tag • 32
- \$InactivityWarning system tag • 32
- \$Language system tag • 32
- \$LogicRunning system tag • 32
- \$Minute system tag • 32
- \$Month system tag • 32
- \$Msec system tag • 32
- \$NewAlarm system tag • 32
- \$ObjHor system tag • 32
- \$ObjVer system tag • 32
- \$Operator system tag • 32
- \$OperatorDomain system tag • 32
- \$OperatorDomainEntered system tag • 32
- \$OperatorName system tag • 32
- \$PasswordEntered system tag • 32
- \$Second system tag • 32
- \$StartDdeConversations system tag • 32
- \$System system tag • 32
- \$Time system tag • 32
- \$VerifiedUserName system tag • 32
- \$Year system tag • 32

•

- .ChartLength dotfield • 163
- .ChartStart dotfield • 164
- .DisplayMode dotfield • 159
- .EngUnits dotfield • 51
- .MaxEU dotfield • 50, 152

- .MaxRange dotfield • 160
- .MaxRaw dotfield • 47
- .MinEU dotfield • 50, 152
- .MinRange dotfield • 159
- .MinRaw dotfield • 47
- .Name dotfield • 115
- .OnMsg dotfield • 53
- .Pen1-8 dotfields • 165
- .Quality dotfield • 95
- .QualityLimit dotfield • 96
- .QualityLimitString dotfield • 97
- .QualityStatus dotfield • 97
- .QualitySubstatus dotfield • 98
- .QualitySubstatusString dotfield • 99
- .RawValue dotfield • 49
- .Reference dotfield • 68
- .ReferenceComplete dotfield • 68
- .ScooterLockLeft dotfield • 167
- .ScooterLockRight dotfield • 168
- .ScooterPosLeft dotfield • 168
- .ScooterPosRight dotfield • 169
- .TagID dotfield • 166
- .TimeDate dotfield • 88
- .TimeDateString dotfield • 88
- .TimeDateTime dotfield • 89
- .TimeDay dotfield • 90
- .TimeHour dotfield • 90
- .TimeMinute dotfield • 91
- .TimeMonth dotfield • 91
- .TimeMsec dotfield • 92
- .TimeSecond dotfield • 92
- .TimeTime dotfield • 93
- .TimeYear dotfield • 94
- .UpdateCount dotfield • 161
- .UpdateInProgress dotfield • 162
- .UpdateInProgress dotfields • 162
- .UpdateTrend dotfield • 163
- .Value dotfield • 52

## A

### Access Names

- application nameIXAccessNamesapplicationname • 61
- assigning to tags • 66
- communication protocolIXAccessNamescommunicationprotocol • 61
- creatingIXAccessNamescreating • 61
- description • 56

- indirect tag remote referencesIXAccessNamesindirecttagremotereferences • 116
- licensing considerations • 211
- polling information • 61
- reinitializing from WindowViewer • 100
- secondary backup serverIXAccessNamessecondarybackupserver • 61
- topic nameIXAccessNamestopicname • 61

#### Archestra Bulk Import Utility

- migrating SuperTags • 127

Average/Bar historical trend • 152

Average/Scatter historical trend • 152

## C

### communication protocols

- DDE • 58

- FastDDE • 58

- NetDDE • 58

- SuiteLink • 58

### Cross Reference Utility

- description • 128

- search for all occurrences • 131

- search for specific occurrences • 131

## D

DDESee Dynamic Data Exchange • 58

DDEStatus topic name • 111

### dotfields

- .DisplayMode dotfield • 159

- .MaxEU dotfield • 50, 152

- .MaxRaw dotfield • 47

- .MinEU dotfield • 152

- .MinRange dotfield • 159

- .Name dotfieldIXdotfieldsNamedotfield • 115

- .OffMsg dotfield • 53

- .Reference dotfield • 68

- .TimeDateString • 88

- description • 38

- listingIXdotfieldslisting • 38

- syntax • 38

- using with local indirect tagsIXdotfieldsusingwithlocalindirecttags • 116

### Dynamic Data Exchange

- communication protocol • 58

HistData itemsIXDynamicDataExchangeHistDataitems • 197

## F

floating-point numbers • 208

ForceLogCurrentValue parameter • 149

functions

HTGetPenName() function • 177

HTSelectTag() function • 175

HTZoomOut() function • 184

IOGetActiveSourceName() function • 110

IOSetAccessName() function • 70

IOSetItem() function • 68

PrintHT() function • 185

## H

HistData

DDE itemsIXHistDataDDEitems • 197

description • 196

historical logging

data collection cycle • 141

description • 141

file naming convention • 141

file retention • 141

historical trends

descriptionIXhistoricaltrendsdescription • 152

HTGetLastError() function • 185

HTGetPenName() function • 177

HTGetTimeAtScooter() function • 178

HTGetTimeStringAtScooter() function • 178

HTGetValue() function • 179

HTGetValueAtScooter() function • 180

HTGetValueAtZone() function • 181

HTScrollLeft() function • 182

HTScrollRight() function • 182

HTSetPenName() function • 176

HTUpdateToCurrentTime() function • 175

HTZoomIn() function • 183

HTZoomOut() function • 184

## I

IEEESee Institute of Electrical and Electronics Engineers • 208

indexed tag names • 24

indirect tags

- concatenating namesIXindirecttagsconcatenatingnames • 115
- descriptionIXindirecttagsdescription • 115
- equating to source tagsIXindirecttagsequatingtosourcetags • 115
- using with local tagsIXindirecttagsusingwithlocaltags • 116
- using with remote referencesIXindirecttagsusingwithremotereferences • 116

Institute of Electrical and Electronics Engineers • 208

InTouchEvent • 61

invalid tag handle • 210

IOGetApplication() function • 67

IOGetNode() function • 67

IOGetTopic() function • 68

IOReinitAccessName() function • 102

IOReinitialize() function • 103

IOStartUninitConversations() function • 103

## L

license

- invalid tag handle • 210

- placeholder tag count • 210

- remote refernce tag count • 211

- system tag count • 210

- tag handles • 210

linear scaling • 64

## M

Message Exchange • 78

Min/Max historical trend • 152

MinEU dotfield • 51

## N

naming conventions for tagsIXnamingconventionsfortags • 24

## R

remote references

- specifying indirect tagsIXremotereferencesspecifyingindirecttags • 116

- using with indirect tagsIXremotereferencesusingwithindirecttags • 116

remote tag countIXremotetagcount • 129

## S

SuiteLink

- description • 58

- diagnosing communication problems • 59

features • 58

## SuperTags

creating a templateIXSuperTagscreatingatemplate • 120

defining instances • 123

deleting a template • 122

description • 119

instances • 119

migrating with ArcestrA Bulk Import Utility • 127

template structure • 119

templates • 119

## system tags

\$Date system tag • 32

\$datestring system tag • 32

\$DateTime system tag • 32

\$Day system tag • 32

\$HistoricalLogging • 149

\$HistoricalLogging system tag • 32

\$Hour system tag • 32

\$InactivityWarning system tag • 32

\$Language system tag • 32

\$LogicRunning system tag • 32

\$Minute system tag • 32

\$Month system tag • 32

\$Msec system tag • 32

\$NewAlarm system tag • 32

\$ObjHor system tag • 32

\$ObjVer system tag • 32

\$Operator system tag • 32

\$OperatorDomain system tag • 32

\$OperatorDomainEntered system tag • 32

\$OperatorName system tag • 32

\$PasswordEntered system tag • 32

\$Second system tag • 32

\$StartDdeConversations system tag • 32

\$System system tag • 32

\$Time system tag • 32

\$TimeString system tag • 32

\$VerifiedUserName system tag • 32

\$Year system tag • 32

license count • 210

## T

### Tagname Dictionary

- creating a new tag • 22
- creating tags overview • 11
- listing SuperTags • 119
- selecting the type of tagIXTagNameDictionaryselectingthetypeoftag • 11
- setting a logging deadband • 26
- setting common tag propertiesIXTagNameDictionarysettingcommontagproperties • 23
- updating tag countsIXTagNameDictionaryupdatingtagcounts • 129

### Tagname dotfields

- .Comment • 54
- .MaxEU dotfield • 50, 51
- .MinRaw dotfield • 47
- .OnMsg dotfield • 53
- .RawValue dotfield • 49
- .Value dotfield • 52

### tags

- adding to a SuperTag instance • 125
- common propertiesIXtagscommonproperties • 23
- configuring for historical loggingIXtagsconfiguringforhistoricallogging • 142
- description • 10
- determining usage • 129
- dynamic remote reference • 211
- handles • 210
- Hist Trend typeIXtagsHistTrendtype • 15
- indirectIXtagsindirect • 115
- initial value • 25
- life cycle • 11
- linear scaling • 64
- maximum EU units • 64
- measurement units • 25
- minimum raw value • 64
- miscellaneous typesIXtagsmiscellaneousypes • 15
- naming conventionsIXtagsnamingconventions • 24
- placeholder count in license • 210
- planning usage • 21
- retentive parameters • 27
- square root scaling • 64
- static remote reference • 211
- Tag ID type • 15



theoretical license maximum • 130

typesIXtagstypes • 11

updating local count • 129

updating local tag countIXtagsupdatinglocaltagcount • 129

updating remote count • 129

Tree view • 71

trends

Average/BarIXtrendsAverageBar • 152

Average/Scatter • 152

description • 151

Min/Max historical trend • 152

typesIXtrendstypes • 151

## V

value deadband

description • 26

setting • 26

Value Time Quality

description • 58

VTQSee Value Time Quality • 58

## W

windows

printing • 31

WindowViewer

description • 11