Wonderware® FactorySuite®
InTouch Reference Guide

User's Guide

Revision A
December, 1997

Wonderware Corporation

100 Technology Drive
Irvine, CA 92618
U.S.A.
(714) 727-3200
http://www.wonderware.com

Trademarks

Contents

Introduction

This reference guide provides an alphabetic reference for each InTouch dot field, windows controls property, alarm object property, system tag and script function. It also includes the add-ons feature of Recipe Manager, SPC Pro and SQL Access Manager. The following describes the contents of the three chapters:

Chapter 1 - "System Tags" containing all InTouch pre-defined system tags.

Chapter 2 - "Dot Fields" containing alarms, distributed alarms, windows controls, historical and tagname dot fields.

Chapter 3 - "InTouch QuickScript Functions" containing built-in InTouch string, math, system, historical, windows controls, distributed alarms, Add-ons (Recipe, SPC Pro, SQL), and miscellaneous functions.

Document Conventions

Examples of Convention	Description
.ChartStart	In general, bold text indicates a .field, system tag or function.
Tagname	In syntax, italic letters indicates placeholders for information you supply.
{ **.HiLimit** | **.HiHiLimit** }	In syntax, braces and a vertical bar indicate a choice between two or more items.
[ErrorMessage=]	Items in square brackets are optional.

To the right of each system tag, property, dot field and script function heading, the usage category (security, alarm, application, etc.) is listed to provide easier referencing. For example:

$AccessLevel **security**

About this Manual

Within this manual you will find several symbols that will assist you while using your online documentation.

When you see a cross reference like this one, it is actually a "hot link" to the referenced section or chapter. Click it to "jump" to that section or chapter. When you jump to another section or chapter and you want to come back to the original section, a "back" option is provided.

These types of cross references indicate that you need to look in another FactorySuite book for more information.

These are "tips" that tell you an easier or quicker way to accomplish a function or task.

To familiarize yourself with the WindowMaker development environment and its tools, see your online *InTouch User's Guide*.

To learn about working with windows, graphic objects, wizards, ActiveX controls and so on, see your online *InTouch User's Guide*.

For details on the runtime environment (WindowViewer), see your online *InTouch Runtime User's Guide*.

The *FactorySuite Systems Administrator's Guide* also provides you with complete information on the other component programs in the suite, system requirements, networking considerations, product integration, technical support, and so on.

Online documentation is included in your FactorySuite software package for all FactorySuite components included in your package. For example, FactorySuite System Administrator's Guide, SPC Pro, SQLAccess Manager, Recipe Manager, IndustrialSQL Sever, InControl and all Wonderware 32-bit I/O Servers. If you purchase FactorySuite+ you also get the online documentation for the InTrack and InBatch components.

Assumptions

This manual assumes you are:

- Familiar with the Windows 95 and/or Windows NT operating system working environment.
- Knowledgeable of how to use of a mouse, Windows menus, select options, and accessing online Help.
- Experienced with a programming or macro language. For best results, you should have an understanding of programming concepts such as variables, statements, functions and methods.

Technical Support

Wonderware Technical Support offers a variety of support options to answer any questions on Wonderware products and their implementation.

Prior to contacting technical support, please refer to the relevant chapter(s) in your *User's Guide* for a possible solution to any problem you may have with using the *Reference Guide*. If you find it necessary to contact technical support for assistance, please have the following information available:

1. Your software serial number.
2. The version of InTouch you are running.
3. The type and version of the operating system you are using. For example, Microsoft Windows NT Version 4.0 workstation.
4. The exact wording of system error messages encountered.
5. Any relevant output listing from the Wonderware Logger, the Microsoft Diagnostic utility (MSD), or any other diagnostic applications.
6. Details of the attempts you made to solve the problem(s) and your results.
7. Details of how to recreate the problem.
8. If known, the Wonderware Technical Support case number assigned to your problem (if this is an on-going problem).

📖 For more information on Technical Support, see your online *FactorySuite System Administrator's Guide*.

Viewing Your FactorySuite License

Your FactorySuite system license information can be viewed through the license viewing utility that is launched from the WindowMaker Help **About** dialog box.

🖱 To access the **About** dialog box, select the **About** command on the WindowMaker **Help** menu.

📖 For more information on the licensing viewing utility, see your *FactorySuite System Administrator's Guide*.

CHAPTER 1

System Tags

InTouch provides several pre-defined system tags. These tags are automatically created by InTouch and can be identified by a preceding dollar sign ($). During runtime, InTouch acts upon the values of these tags in response to system events. System tags can be used anywhere an InTouch tag can be used, such as in animation links of scripts.

$AccessLevel security

Defines the access level of the currently logged-in user.

Usage

```
$AccessLevel
```

Remarks

This is a read-only tag who's value is determined by the AccessLevel assigned to the currently logged-in user's profile within InTouch. This profile can be accessed by selecting the **Configure Users** menu in WindowViewer.

The actual numeric value of **$AccessLevel** has no meaning to WindowViewer. Any "security" desired must be created by you, the designer. By utilizing **$AccessLevel**, many things can be set to enable security within the system.

Data Type

Integer (read only)

Valid Values

0 through 9999

Examples

This statement is used for the visibility link to make an object, such as a button, visible based on the logged on user's access level:

```
$AccessLevel >= 2000;
   Objects can have a "disable" link associated with them, with
   the expression based on $AccessLevel.

$AccessLevel < 5411;

IF $AccessLevel <=500 THEN
   Show "Access Denied"; {popup window denying access}

ELSE
   Show "Access Granted"; {popup window granting access}

ENDIF;
```

See Also

$Operator, $OperatorEntered, $PasswordEntered; $ConfigureUsers

$AlarmLogging alarms

Reinitializes alarm logging and printing during runtime. This tag is equivalent to the **Restart Alarm Log** command on the WindowViewer **Special** menu.

Usage

```
$AlarmLogging = 1;
```

Remarks Setting this value to 1 reinitializes alarm logging and printing during runtime.

Note The **$AlarmLogging** tagname cannot be used to turn off alarm logging or printing. It is strictly used in place of selecting **Restart Alarm Log** on the **Special** menu in WindowViewer. If you display the value of **$AlarmLogging**, it will always show 0 (off) - even when logging is taking place. Setting **$AlarmLogging** to a value other than 1 has no meaning, and the results are undefined.

Data Type Discrete (write only)

Valid Values 1

Example This statement will perform the same action as selecting **Restart Alarm Log** on the **Special** menu in WindowViewer.

```
$AlarmLogging = 1;
```

See Also **$HistoricalLogging**

$AlarmPrinterError alarms

Alarm printer error.

Usage

```
$AlarmPrinterError
```

Remarks Contains a value of 1 if there is a printer error on the alarm printer.

Data Type Discrete (read only)

Valid Values 0 = No error on alarm printer
1 = Error on alarm printer

Example This statement, if used as the expression in an Analog Display Link, will display a value of 1 if there is an error on the Alarm Printer; otherwise, it will display 0.

```
$AlarmPrinterError

IF $AlarmPrinterError == 1 THEN
   DisplayMessageTag = "Error on alarm printer";
ELSE
   DisplayMessageTag = "No error on alarm printer";
ENDIF;
```

$AlarmPrinterNoPaper alarms

Alarm printer is out of paper.

Usage **$AlarmPrinterNoPaper**

Remarks Contains a value of 1 if the alarm printer is out of paper.

Data Type Discrete (read only)

Valid Values 0 = Alarm printer has paper
1 = Alarm printer is out of paper

Example This statement, if used as the expression in an Analog Display Link, will display a value of 1 if the alarm printer is out of paper; otherwise, it will display 0.

```
$AlarmPrinterNoPaper
```

Use in an OnTrue condition action script:

```
Condition:
   $AlarmPrinterOffline == 1

Script:
   show "window";
   playsound("c:\winnt\system32\alarm.wav",1);
```

$AlarmPrinterOffline alarms

Alarm printer is offline.

Usage **$AlarmPrinterOffline**

Remarks Contains a value of 1 if the printer is offline

Data Type Discrete (read only)

Valid Values 0 = Alarm printer is online

1 = Alarm printer is offline

Example This statement, if used as the expression in an Analog Display Link, will display a value of 1 if the alarm printer is offline; otherwise, it will display 0.

```
$AlarmPrinterOffline

IF $AlarmPrinterOffline == 1 THEN
   Call PLCHorn( ); {A Quick function which turns on an audible
alarm on the plant floor}

ENDIF;
```

$AlarmPrinterOverflow alarms

Alarm printer buffer overflow.

Usage `$AlarmPrinterOverflow`

Remarks Contains a value of 1 if there is an alarm printer error (too many characters sent to printer).

Data Type Discrete (read only)

Valid Values 0 = Alarm printer buffer is not full

1 = Alarm printer buffer is full and being overflowed

Example This statement, if used as the expression in an Analog Display Link, will display a value of 1 if the alarm printer buffer is full; otherwise, it will display 0.

```
$AlarmPrinterOverflow
```

$ApplicationChanged application

Signals that the master application has changed in a Network Application Development (NAD) architecture.

Usage `$ApplicationChanged`

Remarks This tag increments every time the update signal is generated by selecting **Notify Clients** on the **Special** menu in WindowViewer. It is reset to 0 when the application is updated. Used for distributed application maintenance.

Data Type Real (read only)

Example The following statement, if used in a Data Change script's tagname field, will cause the body of the script to execute. The script's body could show a window informing the user to restart WindowViewer to have the change take effect.

```
$ApplicationChanged
```

$ApplicationVersion application

Contains the current version number of the application. This number changes each time a tagname or QuickScript is changed, added or deleted.

Usage `$ApplicationVersion`

Remarks None

Data Type Real (read only)

Example Used in an Analog Display Link, this system tag will display the current version of the application that is running within WindowViewer.

`$ApplicationVersion`

$ChangePassword security

Displays the Change Password dialog box. Equal to selecting **Security** then **Change Password** on the **Special** menu in WindowViewer.

Usage `$ChangePassword`

Remarks Setting the value to 1 causes the Change Password dialog box to be displayed. Once the dialog box is closed, InTouch automatically resets the value to 0. Setting this system tag to a value other than 1 has no meaning, and the results are undefined.

Data Type Discrete (write only)

Valid Values 1

Example A discrete pushbutton can be created to allow the user to display the Change Password dialog box. The pushbutton should have a single Discrete Pushbutton link attached to it, and the desired action would be "Set." When pushed, the button would set (make equal to 1) the system tag **$ChangePassword**, causing the dialog to appear.

See Also **$AccessLevel, $OperatorEntered, $PasswordEntered, $Operator, $ConfigureUsers**

$ConfigureUsers security

Displays the generic **Configure Users** dialog box. Equal to selecting **Security** then **Change Password** on the **Special** menu in WindowViewer.

Usage `$ConfigureUsers`

Remarks Setting the value to 1 causes the **Configure Users** dialog box to be displayed. Once the dialog box is closed, InTouch automatically resets the value to 0. The user must have an **$AccessLevel** of >9000 to display this dialog. Setting this system tag to a value other than 1 has no meaning, and the results are undefined.

Data Type Discrete (write only)

Valid Values 1

Example A discrete pushbutton can be created to allow the user to display the **Configure Users** dialog box. The pushbutton should have a single Discrete Pushbutton link attached to it, and the desired action would be "Set." When pushed, the button would set (make equal to 1) the system tag **$ConfigureUsers**, causing the dialog to appear.

See Also **$Operator, $OperatorEntered, $ChangePassword, $PasswordEntered, $AccessLevel**

$Date system

Contains the whole number of days which have passed since 1/1/70.

Usage `$Date`

Remarks None

Data Type Integer (read only)

Example
```
StringFromTime(($Date*86400)+($Time/1000),3); {Returns current time}
```

$DateString system

Contains the date in the same format set in the WIN.INI file.

Usage `$DateString`

Remarks The date format is set through the Windows Control Panel or by double clicking the Clock in the System Tray on the right side of the taskbar.

Data Type Memory message (read only)

Example

```
If StringRight($DateString,2)== "00" THEN
   LogMessage("The Year 2000!");
ENDIF;
```

$DateTime system

Contains the fractional number of days which have passed since 1/1/70.

Usage `$DateTime`

Remarks None

Data Type Real (read only)

Example

```
If StringFromTime($DateTime,4)== "Fri" THEN
   DisplayMessageTag = "It's Friday!";
ENDIF;
```

$Day system

Contains the current day of the month.

Usage `$Day`

Remarks Contains a value between 1 and 31.

Data Type Integer (read only)

Example

```
If $Day == 15 THEN
   Show "Mid-Month Washdown Window";
ENDIF;
```

$HistoricalLogging historical

Monitors and/or controls starting and stopping of historical logging. Equal to selecting **Restart Historical Logging** on the **Special** menu in WindowViewer, or selecting **Stop Historical Logging** on the **Special** menu in WindowViewer.

Usage `$HistoricalLogging`

Remarks Setting this system tag equal to 0 stops historical logging. Setting the value to 1 re-starts historical logging. You cannot "start" historical logging via **$HistoricalLogging** unless logging was previously enabled by selecting **Configure** then **Historical Logging** on the **Special** menu in WindowMaker and is stopped.

Data Type Discrete (read / write)

Example

```
IF (InfoDisk("C",4,$Second)/1024)<50 THEN {Shutdown Historical
logging if free space is less than 50MB}
   $HistoricalLogging = 0;
ENDIF;
```

See Also $AlarmLogging

$Hour system

Contains the current hour of the day.

Usage `$Hour`

Remarks Contains a value between 0 and 23.

Data Type Integer (read only)

Example On True Condition Action Script:

```
$Hour == 20 AND $Second == 30
   PrintWindow("Day Batch Summary",1,1,0,0,0);
```

$InactivityTimeout security

Indicates that the time configured for the automatic log-off of the user has elapsed.

Usage `$InactivityTimeout`

Remarks Contains the value of 1 when automatic log-off has elapsed. To configure the tagname time, on the **Special** menu, point to **Configure** and then, click **WindowViewer** or, in the Application Explorer, under **Configure**, double-click **WindowViewer**. The **WindowViewer Properties** dialog box appears with the **General** properties sheet active. Enter the time value in seconds under the **Inactivity** group.

Data Type Discrete (read only)

See Also **$InactivityWarning**

Example On True Condition Action Script:

```
$InactivityTimeout == 1
Script:
    Show "You have been logged off window";
```

$InactivityWarning security

Indicates that the time configured for warning the user that log-off is about to occur has elapsed.

Usage `$InactivityWarning`

Remarks Contains the value of 1 when log-off is about to occur. The Inactivity timer is reset by mouse clicks or keyboard activity only. To configure the tagname time, on the **Special** menu, point to **Configure** and then, click **WindowViewer** or, in the Application Explorer, under **Configure**, double-click **WindowViewer**. The **WindowViewer Properties** dialog box appears with the **General** properties sheet active. Enter the time value in seconds under the **Inactivity** group.

Data Type Discrete (read only)

Example

```
If $InactivityWarning == 1 THEN
    Show "You are about to be logged off-window";
ENDIF;
```

See Also **$InactivityTimeOut**

$LogicRunning system

Monitors and/or controls the running of scripts. Equal to selecting **Start Logic** on the **Logic** menu in WindowViewer or selecting **Halt Logic** on the **Logic** menu in WindowViewer.

Note Asynchronous User Defined Function scripts that are currently executing cannot be stopped. However, you can prevent new scripts from executing.

Usage `$LogicRunning`

Remarks Setting the value to 1 starts the script logic. Setting the value to 0 stops the script logic.

Data Type Discrete (read / write)

$Minute system

Contains the current minute of the hour.

Usage `$Minute`

Remarks Contains a value between 0 and 59.

Data Type Integer (read only)

Example In an animation link value output string expression field type:

```
IF InfoFile("C:\InTouch.32\WIZ.INI",1,$Minute)==1 THEN
   LogMessage("The File Exists!");
ENDIF;
```

$Month system

Contains the current month of the year.

Usage `$Month`

Remarks Contains a value between 1 and 12.

Data Type Integer (read only)

Example

```
IF $Month==10 THEN
   CurrentMonthName = "October";
ENDIF;
```

$Msec system

Contains the current milliseconds. The rate of update for this system tag is increased by changing the settings for Tick Interval and Update Time Variables. On the **Special** menu, point to **Configure** and then, click **WindowViewer** or, in the Application Explorer under **Configure**, double-click **WindowViewer**. The **WindowViewer Properties** dialog box appears with the **General** properties sheet active.

Usage `$Msec`

Remarks Contains a value between 0 and 999.

Data Type Integer (read only)

$NewAlarm alarms

Indicates that a new alarm has occurred.

Usage `$NewAlarm`

Remarks Contains a value of 1 when a new alarm occurs. This is only set when a new local alarm occurs. It is not set for remote alarms.

Data Type Discrete (read / write)

Example This tagname can be linked to the **PlaySound** logic function, causing an audible alarm to sound. An acknowledgment pushbutton can be created to allow the operator to reset this value to 0 and acknowledge the alarm.

$ObjHor system

Contains the horizontal pixel location of the center of a selected object.

Usage `$ObjHor`

Remarks None

Data Type Integer (read only)

See Also **$ObjVer**

$ObjVer system

Contains the vertical pixel location of the center of a selected object.

Usage **$ObjVer**

Remarks None

Data Type Integer (read only)

See Also **$ObjHor**

$Operator security

Controls the user's ability to perform specific functions.

Usage **$Operator**

Remarks Contains the name of the user currently logged on.

Data Type Message (read only)

Example Access a specific window could be controlled by entering the following script:

```
IF $Operator == "DayShift" THEN
   Show "Control Panel Window";
   ELSE Show "Wrong Operator";

ENDIF;
```

See Also **$OperatorEntered, $AccessLevel, $PasswordEntered, $ChangePassword, $ConfigureUsers**

$OperatorEntered security

Used to enter a valid user name.

Usage `$OperatorEntered`

Remarks Creates a custom log-on window. Touch-sensitive input objects and/or scripts can be linked to this tagname to set the "User Name" for the user.

Note When **$OperatorEntered** is valid, **$AccessLevel** and **$Operator** will be set to their pre-defined values.

Data Type Message (read / write)

See Also **$AccessLevel, $Operator, $PasswordEntered, $ChangePassword, $ConfigureUsers**

$PasswordEntered security

Used to enter a valid password.

Usage `$PasswordEntered`

Remarks This tag will always read back as an empty string. Display links tied to **$PasswordEntered** will always be blank. Since the tag always returns an empty string, data change scripts will never trigger off of **$PasswordEntered**. Can be used to create a custom log-on window. Touch-sensitive input objects and/or scripts can be linked to this tagname to set the "Password" for the user.

Note When **$PasswordEntered** is valid, **$AccessLevel** and **$Operator** will be set to their pre-defined values.

Data Type Message (write only)

See Also **$AccessLevel, $Operator, $OperatorEntered, $ChangePassword, $ConfigureUsers**

$Second system

Contains the current second.

Usage `$Second`

Remarks Contains a value between 0 and 59.

Data Type Integer (read only)

Example
```
Wave=100*Sin(6*$Second);
```

$StartDdeConversations system

Starts uninitiated conversations during runtime when the **Special** menu has been disabled. Equal to selecting **Start Uninitiated Conversations** on the **Special** menu in WindowViewer.

Usage `$StartDdeConversations`

Remarks Setting the value to 1 starts uninitiated DDE conversations.

Data Type Discrete (read / write)

$System alarms

The default alarm group.

Usage `$System`

Remarks If a tagname is not assigned to a specific Alarm Group name, it is automatically assigned to this root group by default. All defined Alarm Groups are descendants of **$System**.

Data Type System Alarm Group (read only)

Example
```
$System.Ack = 1;{Acknowledges All Alarms}
```

$Time system

Contains the time in milliseconds since midnight.

Usage `$Time`

Remarks None

Data Type Integer (read only)

Example

```
Sec_Midnight = $Time/1000;{Number of Seconds since Midnight}
```

$TimeString system

Contains the time in the same format set in the WIN.INI file.

Usage `$TimeString`

Remarks The time format is set through the Windows Control Panel or by double-clicking the clock in the system tray in the lower right corner of the taskbar.

Data Type String (read only)

Example

```
BatchStartString = $TimeString;
```

$Year system

Contains the current year in four digits.

Usage `$Year`

Remarks Contains the year in the following format:

1990

Data Type Integer (read only)

Example

```
CurrentYear = $Year;
NoYrsTill2000 = 2000 - CurrentYear;
```

CHAPTER 2

Dot Fields

InTouch uses dot fields to expose properties of objects like tags, historical trends, and windows controls. These dot fields can be used to monitor and modify those properties. Depending on the object, these properties may be accessed in one of two ways: directly via a *Tagname.field* syntax, or indirectly via a script function.

All objects, except windows controls and the distributed alarm object, use the *Tagname.field* syntax. To access these properties, you simply type the object name followed by the property you want to access in a script or animation link. For example, to allow runtime changes to the HiHi alarm limit on tagname Analog_Tag, an Analog - User Input touch link could be applied to a pushbutton and Analog_Tag.HiHiLimit would be entered as the expression. During runtime, the operator would simply click on this button and type in a new value for the HiHi alarm limit being used for Analog_Tag.

Windows controls and the distributed alarm object use the *GetPropertyX* and *SetPropertyX* script functions, where X is the data type of the property (D = Discrete, I = Integer and M = Message). To avoid confusion with InTouch dot fields the windows controls and distributed alarm objects are listed at the end of this chapter. These functions can also be used in scripts or animation links to access properties.

For more information on the *GetProperty* and *SetProperty* script functions, refer to Chapter 3 in this user's guide.

Tagname Types

When you are defining tagnames in the InTouch database, you must assign a specific type to each tagname according to its usage. For example, if the tagname is to read or write values coming to or from another Windows application such as a I/O Server, it must be a I/O type tagname. The following describes each InTouch tagname type and its usage.

Memory Type Tagnames

Memory tagname types exist internally within your InTouch application. You use them to create system constants and simulations. You can also use them to create calculated variables that are accessed by other Windows programs. For example, you can define a memory tagname with the initial value of 3.1416 or, you could store recipes in groups of memory tagnames. In simulations, you can use memory tagnames to control the actions of a background script. For example, you could define a memory tagname "COUNT" that is changed in an action script to cause various animation effects to occur for the current STEP of a process. There are four Memory types:

Memory Discrete

Internal discrete tagname with a value of either 0 (False, Off) or 1 (True, On).

Memory Integer

A 32-bit signed integer value between -2,147,483,648 and 2,147,483,647.

Memory Real

Floating (decimal) point memory tagname. The floating point value may be between $\pm 3.4e^{38}$. All floating point calculations are performed with 64-bit resolution, but the result is stored in 32-bit.

Memory Message

Text string tagname that can be up to 131 characters long.

I/O Type Tagnames

All tagnames that read or write their values to or from another Windows program are I/O type tagnames. This includes all inputs and outputs from programmable controllers, process computers and data from network nodes. I/O tagnames are accessed either through the Microsoft Dynamic Data Exchange (DDE) protocol or the SuiteLink transport protocol.

When the value of a read/write I/O type tagname changes, it is immediately written to the remote application. The tagname may also be updated from the remote application whenever the item to which the tagname is linked changes in the remote application. By default, all I/O tagnames are set to read/write . However, you can restrict them to read only by selecting the Read Only option in the **Tagname Dictionary** dialog box. There are four I/O types:

I/O Discrete

Discrete input/output tagname with a value of either 0 (False, Off) or 1 (True, On).

I/O Integer

A 32-bit signed integer value between -2,147,483,648 and 2,147,483,647.

I/O Real

Floating (decimal) point tagname. The floating point value may be between $\pm 3.4e^{38}$. All floating point calculations are performed with 64-bit resolution, but the result is stored in 32-bit.

I/O Message

Text string input/output tagname that can be up to 131 characters long.

Indirect Discrete, Indirect Analog, Indirect Message

Indirect type tagnames allow you to create one window and reassign the tagnames in that window to multiple sources. For example, you could create a Data Change script that would modify the source for all tagnames in a window based on a value that has changed.

Miscellaneous Type Tagnames

There are several special tagname types that you can assign to tagnames to perform complex functions such as creating dynamic alarm displays, historical trends, monitoring or controlling the tagname each historical trend pen is plotting. There are also indirect tagname types that you can use to reassign a tagname to multiple sources. These special tagname types are described below.

Group Var

The **Group Var** type is used for a tagname with an assigned Alarm Group to create dynamic alarm displays, disk logs and print logs. You use **Group Var** type tagnames to create alarm windows or alarm logs that display all alarms associated with a specific group variable. You can also control the alarms that are displayed or logged by assigning a different Alarm Group to the **Group Var** tagname.

A **Group Var** type tagname can also be used to create buttons that the operator clicks to selectively display alarms for different areas of a plant in the same alarm window. All of the **dot fields** associated with Alarm Groups can be applied to **Group Var** tagnames.

📖 For more information on Alarms, see your online *InTouch User's Guide*.

Hist Trend

InTouch requires a **Hist Trend** type tagname when you create a historical trend. All of the **dot fields** associated with historical trends can be applied to **Hist Trend** tagnames.

Tag ID

This is a special type that is used with historical trend objects. You use **Tag ID** type tagnames to retrieve information about tagnames being plotted in a historical trend. In most cases, you would use **Tag ID** tagnames to display the name of the tagname assigned to a specific pen or, to change the tagname assigned to the pen.

SuperTags

InTouch SuperTags allow you to define composite tagname types. You can define SuperTags with up to 64 members and 2 nesting levels. Members behave exactly like normal tagnames. They support trending, alarming and all tagname **dot fields**.

📖 For more information on SuperTags see your *InTouch User's Guide*.

Tagname Type vs. Dot Field Usage Matrix

The matrix on the next page illustrates a quick reference between the Tagname Type and the .Dot Field. The footnote below applies to the matrix on the next page.

* Only supported in InTouch 7.0 only. If these memory points are requested as I/O points from a client, View (server) provides its local time stamp, the moment the value is sent to the client. However, from a local standpoint memory tags always have GOOD quality, and NO Timestamp. That is the timestamp is NOT updated based on the memory values changing in scripts or links.

Tagname Types:	Memory				I/O				Indirect			Miscellaneous		
	Discrete	Integer	Real	Message	Discrete	Integer	Real	Message	Discrete	Analog	Message	Group Var	Hist Trend	TagID
Field Names														
.Ack	■	■	■		■	■	■		■	■		■		
.Alarm	■	■	■		■	■	■		■	■		■		
.AlarmDevDeadband		■	■			■	■			■				
.AlarmEnabled	■	■	■		■	■	■		■	■		■		
.AlarmValDeadband		■	■			■	■			■				
.ChartLength													■	
.ChartStart													■	
.Comment	■	■	■	■	■	■	■	■	■	■	■	■	■	■
.DevTarget		■	■			■	■			■				
.DisplayMode													■	
.EngUnits *		■	■			■	■			■				
.HiHiLimit		■	■			■	■			■				
.HiHiStatus		■	■			■	■			■				
.HiLimit		■	■			■	■			■				
.HiStatus		■	■			■	■			■				
.LoLimit		■	■			■	■			■				
.LoLoLimit		■	■			■	■			■				
.LoLoStatus		■	■			■	■			■				
.LoStatus		■	■			■	■			■				
.MajorDevPct		■	■			■	■			■				
.MajorDevStatus		■	■			■	■			■				
.MaxEU		■	■			■	■			■				
.MaxRange													■	
.MaxRaw *					■	■	■	■	■	■	■			
.MinEU		■	■			■	■			■				
.MinorDevPct		■	■			■	■			■				
.MinorDevStatus		■	■			■	■			■				
.MinRange													■	
.MinRaw *					■	■	■	■	■	■	■			
.Name	■	■	■	■	■	■	■	■	■	■	■	■	■	■
.Normal	■	■	■		■	■	■		■	■		■		
.OffMsg *	■				■				■					
.OnMsg *	■				■				■					
.Pen1 through .Pen8													■	
.QualityLimit *	■	■	■	■	■	■	■	■	■	■	■			
.QualityLimitString *	■	■	■	■	■	■	■	■	■	■	■			
.QualityStatus *	■	■	■	■	■	■	■	■	■	■	■			
.QualityStatusString *	■	■	■	■	■	■	■	■	■	■	■			
.QualitySubstatus *	■	■	■	■	■	■	■	■	■	■	■			
.QualitySubstatusString *	■	■	■	■	■	■	■	■	■	■	■			
.RawValue *					■	■	■	■	■	■	■			
.Reference					■	■	■	■	■	■	■			
.ReferenceComplete					■	■	■	■	■	■	■			
.ROCPct		■	■			■	■			■				
.ROCStatus		■	■			■	■			■				
.ScooterLockLeft													■	
.ScooterLockRight													■	
.ScooterPosLeft													■	
.ScooterPosRight													■	
.SPCStatus *		■	■			■	■			■				
.TagID	■	■	■		■	■	■		■	■				
.TimeDate *	■	■	■	■	■	■	■	■	■	■	■			
.TimeDay *	■	■	■	■	■	■	■	■	■	■	■			
.TimeHour *	■	■	■	■	■	■	■	■	■	■	■			
.TimeMinute *	■	■	■	■	■	■	■	■	■	■	■			
.TimeMonth *	■	■	■	■	■	■	■	■	■	■	■			
.TimeMsec *	■	■	■	■	■	■	■	■	■	■	■			
.TimeSecond *	■	■	■	■	■	■	■	■	■	■	■			
.TimeTime *	■	■	■	■	■	■	■	■	■	■	■			
.TimeDateTime *	■	■	■	■	■	■	■	■	■	■	■			
.TimeDateString *	■	■	■	■	■	■	■	■	■	■	■			
.TimeTimeString *	■	■	■	■	■	■	■	■	■	■	■			
.TimeYear *	■	■	■	■	■	■	■	■	■	■	■			
.UnAck	■	■	■		■	■	■		■	■		■		
.UpdateCount													■	
.UpdateInProgress													■	
.UpdateTrend													■	
.Value	■	■	■	■	■	■	■	■	■	■	■	■		■
.00 through .31		■	■			■	■			■				

.Ack alarms

Monitors and/or controls the alarm acknowledgment status of local alarm(s).

Usage

`Tagname.Ack=1`

Parameters	Description
Tagname	Any Discrete, Integer or Real tagname type, Indirect Analog or Group Variable.

Remarks

Set this dot field to a value of 1 to acknowledge any outstanding alarm(s) the specified tagname/group may have associated with it. When the specified tagname is of type **Group Var** or is an Alarm Group, all unacknowledged alarms associated with the tags within the specified group are acknowledged. When the specified tagname is of any type other than Group Var, only the unacknowledged alarm associated with that tagname is acknowledged. Setting this dot field to a value other than 1 has no meaning, and the results are undefined.

Data Type

Discrete (read/write)

Valid Values

1

Examples

The following statement acknowledges an alarm associated with a tagname named "Tag1."

```
Tag1.Ack=1;
```

This next example would be used to acknowledge all unacknowledged alarms within the alarm group named **PumpStation**:

```
PumpStation.Ack=1;
```

An indirect AlarmGroup (using GroupVar) can be used to acknowledge alarms. For example, using an assignment such as:

```
StationAlarms.Name = "PumpStation";
```

Where **StationAlarms** is defined as a GroupVar type tagname and is then associated with **PumpStation**. Thus the statement below is similar to the above examples, except it is used to acknowledge any unacknowledged alarms within the alarm group **PumpStation**, which is currently associated with the GroupVar tag named **StationAlarms**.

```
StationAlarms.Ack=1;
```

Note **.Ack** has an inverse .field called **.Unack**. When an unacknowledged alarm occurs, **.Unack** will be set to **1**. **.Unack** can then be used in animation links or condition scripts to trigger annunciators for any unacknowledged alarms.

See Also

.Alarm and .Unack

.Alarm alarms

Equal to 1 when an alarm condition exists for the specified tagname.

Usage

Tagname`.Alarm`

Parameters	Description
Tagname	Any Discrete, Integer or Real tagname type, Indirect Analog or Group Variable.

Remarks

This read-only dot field is normally equal to 0. When an alarm condition exists for the specified tagname, it is set to a value of 1 by the system. It will remain equal to 1 until the alarm condition no longer exists. If the specified tagname is of type GroupVar or is the name of an Alarm Group, the .Alarm field will be set equal to 1 if any of the tags within that group are in alarm. .Alarm has an inverse dot field called **.Normal**.

Data Type

Discrete (read-only)

Valid Values

0 or 1 (discrete)

Examples

The following statement verifies if "Tag1" has an active alarm associated with it:

```
IF (Tag1.Alarm == 1) THEN
```

The body of this IF-THEN statement would be processed if there were active alarms within the alarm group named **PumpStation**.

```
IF (PumpStation.Alarm == 1) THEN
   MyAlarmMessage="The pumping station currently has an ALARM!";
ENDIF;
```

This dot field is not linked to the **.Ack** or **.Unack** dot fields. Therefore, even when an active alarm has been acknowledged, this dot field remains equal to 1.

See Also

.Ack, .Normal, Ack()

.AlarmDevDeadband alarms

Monitors and/or controls the deviation percentage deadband for both minor and major deviation alarms.

Usage *Tagname*`.AlarmDevDeadband`

Parameters	Description
Tagname	Any Integer, Real of Indirect Analog Tag Type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values 0 through 100 (integer)

Example The following statement changes the deviation deadband percentage to 25%:

```
Tagname.AlarmDevDeadband=25;
```

See Also **.AlarmValDeadband**

.AlarmEnabled alarms

Disables and/or enables events and alarms for a tagname.

Usage *Tagname*.**AlarmEnabled**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog or Group variable.

Remarks When **.AlarmEnabled** is set equal to 0, all events and alarms are ignored. They are not stored in a buffer, nor are they written to disk. It is very important to re-enable events/alarms, whenever possible, to avoid data loss.

Data Type Discrete (read/write)

Valid Values 0 = Disable Alarms and Events

1 = Enable Alarms and Events (*default*)

Examples The following statement disables all events and alarms associated with the tag named "Tag1."

```
Tag1.AlarmEnabled=0;
```

The body of the following IF-THEN statement will be processed if the events and alarms are enabled for the AlarmGroup named "PumpStation."

```
IF (PumpStation.AlarmEnabled == 1) THEN
   MyAlarmMessage="The Events and Alarms for the Pump
   Station are enabled"
ENDIF;
```

See Also **.Alarm**

.AlarmValDeadband alarms

Monitors and/or controls the value of an alarm's deadband.

Usage *Tagname*`.AlarmValDeadband`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in the value range configured for the specified tagname.

Example The following statement changes the deadband of a tag named "Tag1" to a value of 25:

```
Tag1.AlarmValDeadband=25;
```

See Also **.AlarmDevDeadband**

.ChartLength historical

Controls the length (span in seconds) of time displayed in a Historical Trend Chart.

Usage *Tagname*.**ChartLength**

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks This read/write dot field is used to set (or verify) the value of the Chart Length of a Historical Trend Chart. The Chart Length is expressed in seconds. The length is defined as the amount of time currently displayed on the Historical Trend Chart. More specifically, the calculation used to retrieve 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);
```

Since Date/Time Stamps are expressed in seconds since midnight on 1/1/70, the calculation results in "seconds of time displayed between the left and right hand sides of the chart."

Whenever adding or subtracting from **.ChartLength**, it is important to remember that you are dealing in seconds. Therefore, if you wish to subtract "2 hours" from the current **.ChartLength**, you need to convert "2 hours" to seconds before performing the calculation, i.e.: (2 hours) * (60 minutes/hour) * (60 seconds/minute) = 7200 seconds.

Data Type Integer (read/write)

Valid Values Any positive integer

Example The following statement forces the chart span to be 1 hour:

```
HtTagname.ChartLength=3600 {60 minutes * 60 seconds/minute};
```

The following statement scrolls the chart left by half:

```
HtTagname.ChartStart=HtTagname.ChartStart -
HtTagname.ChartLength / 2;
```

The following statement scrolls the chart left by 10%:

```
HtTagname.ChartStart=HtTagname.ChartStart - (.10 *
   HtTagname.ChartLength);
```

See Also **.ChartStart**

.ChartStart historical

Controls the starting date/time stamp of a Historical Trend Chart.

Usage *Tagname*`.ChartStart`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks This read/write dot field is used to set (or verify) the value of the starting (left hand side) date/time stamp of a Historical Trend Chart. The **.ChartStart** field is expressed as the number of seconds since 12:00AM, 1/1/70. The "start" is defined as the first date/time stamp on the left hand side of the Historical Trend Chart.

Data Type Integer (read/write)

Valid Values Any positive integer

Example The following statement scrolls the chart to the right by 1 minute:

```
HtTagname.ChartStart=HtTagname.ChartStart + 60;
```

.Comment tagname

Contains the value of the comment field entered for a tagname in the Tagname Dictionary.

Usage *Tagname*`.Comment`

Parameters	Description
Tagname	Any tagname.

Remarks While this field is read only in WindowMaker, the value can be changed in View (memory only). Once View is shut down and restarted again, the old value for the comment field is used. This field is used by the distributed alarm system for alarm comments.

Data Type Message (read only)

Valid Values Any string containing from 1 to 131 characters. See note below for character string limitations.

Note The comment field for a tag can only be changed while in the Tagname Dictionary. Inputting a value into the **.Comment** field while in WindowViewer, will not write to the Tagname Database. While a tag comment allows 131 characters in the Tagname Database, the Runtime database imposes a 50 character limit.

Example The following statement assembles a string (contents of a message tagname) by containing the Name of the tagname and the tagname's Comment field:

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

Note This field writes to the runtime database. It is not saved in the tagname database.

.DevTarget alarms

Monitors and/or controls the target for minor and major deviation alarms.

Usage *Tagname*.**DevTarget**

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in the value range configured for the specified tagname.

Example The following statement sets the Deviation Target for the tag named **MyTag** to a value of 500:

```
MyTag.DevTarget=500;
```

.DisplayMode historical

Determines the method to be used in displaying values on the trend.

Usage *Tagname*.**DisplayMode**

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks None

Data Type Analog (read/write)

Valid Values 1 = Displays the min/max for each sample period (default).

2 = Displays the average for each sample period in a "scatter" diagram.

3 = Displays the average for each sample period in a "bar chart" diagram.

Example The following statement instructs the Historical Trend associated with **MyHistTrendTag** to display a bar chart type diagram:

```
MyHistTrendTag.DisplayMode=3;
```

.EngUnits tagname

The engineering units field allows the user to access the engineering units of an analog tagname as specified in the tagname dictionary.

Usage *Tagname*`.EngUnits`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks This field is a text field and does not effect scale, conversions, or format of the Tagname value.

Data Type Message (read / write)

Valid Values Any string containing from 0 to 31 characters.

Example

```
IF Temperature.EngUnits == "Celsius" THEN
   CALL TempConvert(Temperature);{A Quick Function which might be
used to convert Celsius to Fahrenheit}
ENDIF;
```

.HiHiLimit alarms

Monitors and/or controls the HiHi Limit for value alarm checks.

Usage *Tagname*`.HiHiLimit`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in value range configured for the specified tagname.

Example The following statement increases the HiHiLimit (alarm limit) of the tag named **MyTag** by a value of 5:

```
MyTag.HiHiLimit=MyTag.HiHiLimit + 5;
```

See Also **.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiLimit, .HiStatus, .HiHiStatus**

.HiHiStatus alarms

Determines whether a HiHi limit alarm exists.

Usage

Tagname`.HiHiStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

This read-only dot field is normally equal to 0. When a HiHi value alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists. This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

Data Type

Discrete (read-only)

Valid Values

0 = Specified Alarm Condition is not present

1 = Specified Alarm Condition is present

Example

The following IF-THEN statement will be processed only when the **.HiHiStatus** ("High-High Alarm") of the tag named **MyTag** is equal to a value of 1. When MyTag goes into High-High Alarm, the body of the IF-THEN will execute:

```
IF (MyTag.HiHiStatus == 1) THEN
OperatorMessage="MyTag has gone into High-High Alarm";

ENDIF;
```

See Also

.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiLimit, .HiHiLimit, HiStatus

.HiLimit alarms

Monitors and/or controls the Hi Limit for value alarm checks.

Usage *Tagname*`.HiLimit`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in value range configured for the specified tagname.

Example
```
Temperature.HiLimit = 212;
```

See Also **.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiHiLimit, HiStatus, .HiHiStatus**

.HiStatus alarms

Determines whether a Hi limit alarm exists.

Usage *Tagname*`.HiStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks This read-only dot field is normally equal to 0. When a Hi value alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists. This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

Data Type Discrete (read-only)

Valid Values 0 = Specified Alarm Condition is not present

1 = Specified Alarm Condition is present

Example
```
IF (MotorAmps.HiStatus == 1) THEN
   CALL PumpShutdown( ); {Quick Function which might be used to
kill a pump motor output}
ENDIF;
```

See Also **.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiLimit, .HiHiLimit, .HiHiStatus**

.LoLimit alarms

Monitors and/or controls the Low Limit for value alarm checks.

Usage *Tagname*`.LoLimit`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in value range configured for the specified tagname.

Example This statement decreases the LoLimit (alarm limit) of the tag named **MyTag** by a value of 10:

```
MyTag.LoLimit=MyTag.LoLimit - 10;
```

See Also **.Alarm, .Ack, .LoLoLimit, .LoStatus, .HiLimit, .HiHiLimit, HiStatus, .HiHiStatus**

.LoLoLimit alarms

Monitors and/or controls the LoLo Limit for value alarm checks.

Usage *Tagname*`.LoLoLimit`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type Analog (read/write)

Valid Values Must be in value range configured for the specified tagname.

Example This statement decreases the LoLoLimit (alarm limit) of the tag named **MyTag** by a value of 10:

```
MyTag.LoLoLimit=MyTag.LoLoLimit - 10;
```

See Also **.Alarm, .Ack, .LoLimit, .HiLimit, .HiHiLimit, HiStatus, .HiHiStatus**

.LoLoStatus alarms

Determines whether a LoLo limit alarm exists.

Usage

Tagname`.LoLoStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

This read-only dot field is normally equal to 0. When a LoLo value alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists. This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

Data Type

Discrete (read-only)

Valid Values

0 = Specified Alarm Condition is not Present

1 = Specified Alarm Condition is Present

Example

The following IF-THEN statement will be processed only when the **.LoLoStatus** ("*Low-Low Alarm*") of the tag named **MyTag** is equal to a value of 1. When MyTag goes into Low-Low Alarm, the body of the IF-THEN will execute:

```
IF (MyTag.LoLoStatus == 1) THEN
OperatorMessage="MyTag has gone into Low-Low Alarm";

ENDIF;
```

See Also

.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiLimit, .HiHiLimit, HiStatus, .HiHiStatus

.LoStatus alarms

Determines whether a Lo limit alarm exists.

Usage

Tagname`.LoStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

This read-only dot field is normally equal to 0. When a Lo value alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists. This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

Data Type

Discrete (read-only)

Valid Values

0 = Specified Alarm Condition is not Present

1 = Specified Alarm Condition is Present

Example

```
IF (WaterLevelTank1.LoStatus == 1) THEN
   PumpShutdown = 1;
   WaterFillValue = 1;
ENDIF;
```

See Also

.Alarm, .Ack, .LoLimit, .LoLoLimit, .HiLimit, .HiHiLimit, HiStatus, .HiHiStatus

.MajorDevPct alarms

Monitors and/or controls the major percentage of deviation for alarm checking.

Usage

Tagname`.MajorDevPct`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type

Integer (read/write)

Valid Values

0 to 100%

Example

The following statement sets the Major Deviation limit property of the tag named **MyTag** to a value of 25%:

```
MyTag.MajorDevPct=25;
```

See Also

.MajorDevStatus

.MajorDevStatus alarms

Determines whether a major deviation alarm exists for the specified tagname.

Usage *Tagname*`.MajorDevStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks This read-only dot field is normally equal to 0. When an alarm condition exists for the specified tagname, the associated dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists.

Data Type Discrete (read-only)

Valid Values 0 = Specified Alarm Condition is not Present

1 = Specified Alarm Condition is Present

Example The following IF-THEN statement will be processed only when the .MajorDevStatus ("Major deviation Alarm") of the tag named **MyTag** is equal to a value of 1. In other words, when MyTag goes into Major Deviation Alarm, the body of the IF-THEN will execute:

```
IF (MyTag.MajorDevStatus == 1) THEN
OperatorMessage="MyTag has gone into a Major Deviation Alarm";
ENDIF;
```

Remarks This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

See Also **.MajorDevPct**

.MaxEU tagname

The maximum values (in engineering units) for the specified tagname.

Usage

Tagname`.MaxEU`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

A value from a server is considered RAW when it first arrives in WindowViewer. These raw values may require scaling. .MinEU and .MaxEU are used to scale RAW values. As a typical example, assume a level gauge is being read by a Programmable Logic Controller in the field. The level transmitter sends back a signal that ranges between 4 and 20mA. The PLC in the field converts this signal (through a digital to analog converter) to an integer value between 0 and 4096. This value is being read into the tag named "TankTwoLevel."

Displaying this RAW value (between 0 and 4096) will present no useful data to the operator. It is therefore necessary to scale this value to an appropriate engineering range. To accomplish this, the Minimum Engineering Units and Maximum Engineering Units fields must be setup correctly. In our example, if the RAW value of 0 (4mA from the field) translated to "0 Gallons" and the value of 4096 (20mA from the field) translated to "100 Gallons", the following setup would be required to display the correct value on the screen:

```
Minimum Raw=0,Maximum Raw z= 4096
```

```
Minimum Engineering Units=0,Maximum Engineering Units=100
```

With these settings, when the RAW value in the field is 4096, the value displayed on the screen will be 100.

Data Type

Real (read-only)

Valid Values

Depends on the type of tagname specified.

Example

Since this dot field is read-only, assigning a value to it is not possible. Displaying the **.MinEU** and **.MaxEU** or using them in calculations is very useful, and can enhance the understanding of the operator by providing basis for your calculations on the screen.

```
DialogValueEntry ("IO_Point_717", IO_Point_717.MinEU,
IO_Point_717.MaxEU, "Please Enter a New Value:");
```

See Also

.MinEU, .MinRaw, .MaxRaw, .RawValue

.MaxRange historical

Represents the percentage of the tagname's Engineering Unit range that should be displayed for each tagname being trended.

Usage *Tagname*`.MaxRange`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks Since many different tags can be displayed on the historical trend at any one time, specifying the **.MinRange** and **.MaxRange** in engineering units would be impossible, since each of the tags could (and usually do) have completely different engineering ranges. Therefore, the min and max are expressed as a percentage of the engineering range of each tagname. This way, regardless of the tagname's true engineering range, the historical trend will simply display the indicated *percentage* of that tagname's particular engineering range.

Data Type Real (read/write)

Valid Values The limits for **.MaxRange** and **.MinRange** are from 0 to 100. **.MinRange** and **.MaxRange** are expressed in percent. .**MinRange** must be less than **.MaxRange**. If a value less than 0 or greater than 100 is assigned to either of these fields, the value will be clamped at 0 or 100. If **.MinRange** is greater than or equal to **.MaxRange**, the trend will not display any data.

Example The following statement sets the maximum range percentage of the Historical Trend associated with the Hist Trend tag named "MyHistTrendTag" to a value of 25%:

```
MyHistTrendTag.MaxRange=25;
```

See Also **.ChartStart, .ChartLength, .DisplayMode, .MinEU, .MaxEU**

.MaxRaw tagname

The high clamp setting for the actual raw value received from an I/O Server by WindowViewer as a client. The value for .MaxRaw field comes from the Max Raw value field in the tagname database for the specified I/O tagname. Any raw value which exceeds this setting is clamped to this value.

Usage

tagname.**MaxRaw**

Parameters	Description
Tagname	Any I/O Discrete, Indirect Discrete, I/O Integer, Memory Real, Indirect Analog, I/O Message and Indirect Message tagname type.

Remarks

This read-only dot field is used to display the Max Raw high clamp setting.

Data Type

Real or Integer (read-only)

Valid Values

Any analog value.

Example

The following could be used to determine if a tagname is out of normal operating range and thus its value was clamped.

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

See Also

.MinRaw, .RawValue, .MinEU, .MaxEU

.MinEU tagname

The minimum values (in engineering units) for the specified tagname.

Usage *Tagname*`.MinEU`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks A value from a server is considered RAW when it first arrives in WindowViewer. These raw values may require scaling. .MinEU and .MaxEU are used to scale RAW values. As a typical example, assume a level gauge is being read by a Programmable Logic Controller in the field. The level transmitter sends back a signal that ranges between 4 and 20mA. The PLC in the field converts this signal (through a digital to analog converter) to an integer value between 0 and 4096. This value is being read into the tag named "TankTwoLevel."

Displaying this RAW value (between 0 and 4096) will present no useful data to the operator. It is therefore necessary to scale this value to an appropriate engineering range. To accomplish this, the Minimum Engineering Units and Maximum Engineering Units fields must be setup correctly. In our example, if the RAW value of 0 (4mA from the field) translated to "0 Gallons" and the value of 4096 (20mA from the field) translated to "100 Gallons", the following setup would be required to display the correct value on the screen:

```
Minimum Raw=0,Maximum Raw = 4096
Minimum Engineering Units=0,Maximum Engineering Units=100
```

With these settings, when the RAW value in the field is 4096, the value displayed on the screen will be 100.

Data Type Real (read-only)

Valid Values Depends on the type of tagname specified.

Example

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

See Also **.MaxEU, .MinRaw, .MaxRaw, .RawValue**

.MinorDevPct alarms

Monitors and/or controls the minor percent of deviation for alarm checking.

Usage

Tagname`.MinorDevPct`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks

Checking Retentive Parameters in the Tagname Dictionary will automatically save any changes made to this field through animation or scripting in WindowViewer.

Data Type

Integer (read/write)

Valid Values

0 to 100%

Example

The following statement sets the Minor Deviation limit property of the tag named **MyTag** to a value of 25%:

```
MyTag.MinorDevPct=25;
```

See Also

.MinorDevStatus

.MinorDevStatus alarms

Determines whether a minor deviation alarm exists for the specified tagname.

Usage *Tagname*`.MinorDevStatus`

Parameters	Description
Tagname	Any Integer, Real or Indirect Analog tagname type.

Remarks This read-only dot field is normally equal to 0. When an alarm condition exists for the specified tagname, the associated dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists.

Data Type Discrete (read-only)

Valid Values 0 = Specified Alarm Condition is not present

1 = Specified Alarm Condition is present

Example The following IF-THEN statement will be processed only when the .MinorDevStatus ("Minor Deviation Alarm") of the tagname named **MyTag** is equal to a value of 1. In other words, when MyTag goes into Minor Deviation Alarm, the body of the IF-THEN will execute.

```
IF (MyTag.MinorDevStatus == 1) THEN
OperatorMessage="MyTag has gone into a Minor Deviation Alarm";
ENDIF;
```

Remarks This dot field is often used in conjunction with the .**Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

See Also **.MinorDevPct**

.MinRange historical

Represents the percentage of a historical trend's range that should be displayed for each tagname being trended.

Usage *Tagname*`.MinRange`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks Since many different tags can be displayed on the historical trend at any one time, specifying the **.MinRange** and **.MaxRange** in engineering units would be impossible, since each of the tags could (and usually do) have completely different engineering ranges. Therefore, the min and max are expressed as a percentage of the engineering range of each tagname. This way, regardless of the tagname's true engineering range, the historical trend will simply display the indicated *percentage* of that tagname's particular engineering range.

Data Type Real (read/write)

Valid Values The limits for **.MaxRange** and **.MinRange** are from 0 to 100. **.MinRange** and **.MaxRange** are expressed in percent. .**MinRange** must be less than **.MaxRange**. If a value less than 0 or greater than 100 is assigned to either of these fields, the value will be clamped at 0 or 100. If **.MinRange** is greater than or equal to **.MaxRange**, the trend will not display any data.

Example

```
TotalChartHeight=MyHistTrendTag.MaxRange -
MyHistTrendTag.MinRange;
```

See Also **.ChartStart, .ChartLength, .DisplayMode, .MinEU, .MaxEU, .MaxRange**

.MinRaw tagname

The low clamp setting for the actual raw value received from an I/O Server by WindowViewer as a client. The value for .MinRaw field comes from the Min Raw value field in tagname database for the specified I/O tagname. Any raw value which falls below this setting is clamped to this value.

Usage *tagname*`.MinRaw`

Parameters	Description
Tagname	Any I/O Discrete, Indirect Discrete, I/O Integer, Memory Real, Indirect Analog, I/O Message and Indirect Message tagname type.

Remarks This read-only dot field is used to display the Min Raw high clamp setting.

Data Type Real or Integer (read-only)

Valid Values Any analog value.

Example The following could be used to determine if a tagname is out of normal operating range and thus its value was clamped.

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

See Also **.MaxRaw, .RawValue, .MinEU, .MaxEU**

.Name tagname

Contains the name of the specified tagname in "String" form.

Usage

Tagname.**Name**

Parameters	Description
Tagname	Any tagname type

Data Type

Message (read/write)

Valid Values

Assignments can only be made to the .Name field of Indirect type tags which includes Group Var and Tag ID. When making assignments to this field, the format of the assignment string must follow tagname naming rules, such as what symbols are appropriate and the first character should always be alphabetical.

Remarks

This field is read-only for the following tagname types:

Hist Trend, Any Memory Type, Any I/O Type

This field is read/write for the following tagname types:

Group Var, Tag ID, Any Indirect Type

In the read-only implementation, this field is useful for specifying the tagname without having to put the tagname in quotation marks, i.e., MyTag.Name is the same as "MyTag." This is highly beneficial when you select **Update Use Counts** on the **Special** menu in WindowMaker or Cross Reference is used to scan for a tagname's usage it cannot see the tagname being used if it is within quotation marks. Therefore, if a tagname's only usage is being specified between "quotes", and you perform an **Update Use Counts** followed by a **Special**, **Delete Unused Tags**, the tagname will be removed, since it is considered, unused. A "" can be used to de-reference an indirect tagname, thus "nullifying" any previous assignment.

In the read/write implementation, this field is even more useful. It allows you to modify the identity of the specified Indirect tagname. When the .Name field of an Indirect tagname is assigned the name of another tagname, assuming both tags are of the same type, the indirect tagname actually becomes the other tagname, and it can be used anywhere in InTouch that the original tagname can be used. Similarly, you can change the Alarm Group that a Group Var type tagname is emulating.

Example

The following read-only statement use of .Name concatenation is used to create a message for the operator:

```
MyMessageTag="The tag named  " + MyTag.Name + " has a
comment of " + MyTag.Comment;
```

The following read/write statement changes the Alarm Group to which the Group Variable type tag named "MyGroupVarTag" is referring:

```
MyGroupVarTag.Name="SomeAlarmGroup";
```

Scenario: You develop a graphic window to display three vital statistics about an oil well, but you have 100 oil wells. The following Data Change Script can be used to update a single graphic window no matter which oil well the operator selects. The tagname for the Data Change Action Script is OilWellNumber.

```
IndOilWellPump.Name = "OilWellPump" + Text(OilWellNumber, "#");
IndOilWellTEP.Name = "OilWellTemp" + Text(OilWellNumber, "#");
IndOilWellPressure.Name = "OilWellPressure" + Text(OilWellNumber,
"#");
```

To force the script to execute use a pushbutton with a touch input Analog Link, using OilwellNumber in the tagname field. In Runtime, when the operator clicks the button and enters a well number the script will execute.

.Normal alarms

.Normal is equal to 1 when there are no alarms for the specified tagname.

Usage

Tagname`.Normal`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Group Variable tagname type.

Remarks

This dot field is used within scripts or display links to indicate that the specified tagname is in a "normal" state (no alarms active). If one or more alarms become active for the specified tagname, this dot field is set equal to zero.

Data Type

Discrete (read-only)

Valid Values

0 = One or more alarms active for the specified tagname

1 = No alarms are active for the specified tagname (default)

Example

The "THEN" body of the following IF-THEN statement will execute whenever there aren't any alarms associated with the tag named "Tag1." When there is one or more alarms active for "Tag1", the "ELSE" body will execute:

```
IF (Tag1.Normal==1) THEN
MyOperatorMessage="Tag1 is OK - No alarms associated
with it";

ELSE
MyOperatorMessage="Tag1 has one or more alarms
active!";
ENDIF;
```

See Also

.Alarm

.OffMsg tagname

The .OffMsg field allows the user to access the off message of a discrete tagname as specified in the tagname dictionary.

Usage *Tagname*`.OffMsg`

Parameters	Description
Tagname	Any Discrete tagname type.

Data Type Message (read/write)

Valid Values Any string containing from 0 to 15 characters.

Example If MyDiscrete has a value of 0, the following script will assign the OffMsg string contained in the tagname database for MyDiscrete, to the tagname StateMessage.

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

See Also **.OnMsg**

.OnMsg tagname

The .OnMsg field allows the user to access the on message of a discrete tagname as specified in the tagname dictionary.

Usage *Tagname*`.OnMsg`

Parameters	Description
Tagname	Any discrete tagname type.

Data Type Message (read/write)

Valid Values Any string containing from 0 to 15 characters.

Example

```
IF IndAnalog.OnMsg == "Running" THEN
   TypeOfTag = "IndAnalog was assigned a Motor Starter";
ENDIF
```

See Also **.OffMsg**

.Pen1 - .Pen8 historical

Controls the tagname being historically trended by each pen.

Usage

```
Tagname{.Pen1 | .Pen2 | .Pen3 | .Pen4 | .Pen5 | .Pen6 | .Pen7 |
.Pen8};
```

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

Due to the complexity of using the **.Pen*X*** fields, it is recommended that the **HTSetPenName** and **HTGetPenName** functions be used, if possible.

Note Only local tags can be assigned to a **.Pen*X*** field. The "provider.tag" notation cannot be used. The "provider.tag" can only be used with **HTSetPenName**.

A good reference to use when learning how these dot fields function is a Historical Trend Wizard placed on the screen and broken apart. This will show how the dot fields were intended to be used.

Data Type

TagID (read/write)

Valid Values

This dot field is type **.TagID**. This means that the "handle" of a tagname may be assigned to this dot field. You cannot assign the name of a tagname to this field. You must assign the associated **.TagID** dot field of a tagname to this dot field. You may want to refer to the description of the **.TagID** dot field for a clearer understanding of this explanation. In general, a TagID type tagname can only be equated to another TagID tagname. It cannot be mixed with any other tagname type unless the .TagID field extension is added to the other tagname.

Although this dot field is considered read/write, its value cannot be directly displayed on the screen.

Examples

The following statement will be used to assign a new tagname to Pen1 of the historical trend associated with the Historical Trend type tagname named "MyHistTrendTag." By processing the following statement within a script Pen1 of the historical trend associated with "MyHistTrendTag" will begin displaying the historically logged data for the tag named "MyLoggedTag." Note how we must append the **.TagID** dot field to the name of the "MyLoggedTag" in order to assign it to Pen1. This is because the . "Pen1" dot field is of "TagID" type and since you must match types when performing assignments (i.e.: DiscreteTag = DiscreteTag, is acceptable, whereas DiscreteTag = MessageTag is illegal and actually makes no sense!) we must assign a TagID type object to that dot field.

```
MyHistTrendTag.Pen1=MyLoggedTag.TagID;
```

Working from the example above, what if we wanted to display the NAME of the tagname we just assigned to "MyHistTrendTag.Pen1"? This would be very useful information for the operator. Additionally it is nice to have the tagname's displayed in a legend near the historical trend for reference. To accomplish this, it would seem natural to display the value of "MyHistTrendTag.Pen1" in a MessageDisplay link, but as you can see if you try this for yourself, it doesn't work. Why not? The actual *value* of the .Pen1 field is an integer that represents a memory location within Window Viewer; not particularly useful for display purposes. To get around this, we need to exercise a little caveat. First, create a new tagname of type TagID; call it "Pen01." Place the following statement underneath the statement from the previous example:

```
Pen01=MyHistTrendTag.Pen1;
```

In the first example, we assigned the tagname "MyLoggedTag" to Pen1 of MyHistTrendTag." In this example we went one step further by assigning the value of Pen1 of "MyHistTrendTag", which is now the TagID of "MyLoggedTag", to the tagname Pen01. Why did we do this? The reason is that we needed a way to get to the ."Name" dot field. We could not simply display "MyHistTrendTag.Name" on the screen because it would always display "MyHistTrendTag." If we then tried to display "MyHistTrendTag.Pen1" on the screen, the link editor wouldn't let us, because Pen1 is of type TagID, which cannot be displayed. So we need a helper-tag to bridge the gap between the display system and the historical system. By assigning the value of "MyHistTrendTag.Pen1" to the TagID type tagname "Pen01" we have essentially assigned "MyLoggedTag" to "Pen01." In fact, we could have accomplished the same task as we did above with the following line of code. They are functionally identical:

```
Pen01=MyLoggedTag.TagID;
```

So, why did we choose the first statement over the second? Because the first is more generic. In addition, you can simply place "MyHistTrendTag.Pen1" as the "tagname" in a data change script and place: Pen01 = MyHistTrendTag.Pen1; in the body. This way, without ever knowing which tagname was just assigned to Pen1 you can assign that same tagname to Pen01. This allows you to constantly display the NAME of the tagname that is assigned to Pen1 of the historical trend on the screen, even when someone changes the tagname either through the regular user interface (click on the trend in VIEW) or through a script somewhere in the system.

Taking this example one step further, you may at some point, wish to display the minimum and maximum Engineering Units of a particular tagname that is being trended. The Historical Trend Wizard does a very nice job of this. We suggest you take a moment and place a Hist Trend Wizard on the screen, break it apart and see how it was constructed. This will give you a good idea of how these dot fields function.

Because TagID type tags, like "Pen01", do not have engineering units or any of the other dot fields normally available to a tagname, we cannot use it to display the engineering units of a tagname assigned to it. For example, we cannot display "Pen01.MaxEU" since the link editor would complain that .MaxEU is not a valid dot field for Pen01 which is of type TagID.

To get around this, we merely need a helper tagname to handle the conversion between TagID type and the display system. First, create a tagname of type Indirect Analog for example, "IndirectAnalogPen1." Within the same data change script as example #2, place the following statement:

```
IndirectAnalogPen1.Name=Pen01.Name;
```

Pen01's name (at this point in our example it should equal "MyLoggedTag") is assigned to the indirect tagname "IndirectAnalogPen1." With this done, we can access all the dot fields normally available to us with respect to an analog tagname in this case the integer tagname called MyLoggedTag. So we can now place the following with an Analog Display Link on the screen:

```
IndirectAnalogPen1.MaxEU
```

This will display the Maximum Engineering units of the tagname currently assigned to the tagname "IndirectAnalogPen1." Since we assigned Pen01.Name to it in the data change script above, we can be certain that it will display the Maximum Engineering units of the tagname currently assigned to Pen1 of "MyHistTrendTag."

See Also **.TagID, HTGetPenName( ), HTSetPenName( )**

.Quality tagname

To completely understand the Quality dot fields used by Wonderware a brief definition of quality standard is stated. The Wonderware Data Quality standard is based on the OLE for Process Control (OPC) proposed quality, which in turn is based on Fieldbus Data Quality Specifications.

Quality flags represent the quality state for an item's data value. This design makes it fairly easy for both Servers and Client applications to determine how much functionality they want to implement.

The low 8 bits (Least Significant Byte) of the Quality flags are currently defined in the form of three bit fields; Quality, Substatus and Limit status arranged as follows:

QQSSSSLL

The Quality field allows the user to access the quality of an I/O tagname as provided by an I/O Server.

Note If the I/O connection becomes invalid, the quality dot fields are automatically reset to the initial value of zero. The **.ReferenceComplete** flag is also set to zero at this time.

Usage

Tagname`.Quality`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Data Type Message (read/write)

Valid Values Value may be 0 to 255

Example

```
IF I0Tag.Quality <> 192 THEN
   LogMessage("This data is not Good!");
ENDIF;
```

Wonderware I/O Servers can report six (6) mutually exclusive states of quality of data being sent back to their clients. They are as follows:

		OPC Quality Bits		OPC QualityBytes .QualityStatus		
		MSByte	LSByte		.QualitySubStatus	
	Hex Value	xxxxxxxx	QQSSSSLL			.QualiyLimit
1. Good	0x00C0	00000000	00000000	Q=3	S=0	L=0
2. Clamped High	0x0056	00000000	00000000	Q=1	S=5	L=2
3. Clamped Low	0x0055	00000000	00000000	Q=1	S=5	L=1
4. Cannot Convert	0x0040	00000000	00000000	Q=1	S=0	L=0
5. Cannot Access Point	0x0004	00000000	00000000	Q=0	S=1	L=0
6. Communications Failed	0x0018	00000000	00000000	Q=0	S=6	L=0

When the client application is unable to communicate with the Server .QualityStatus is 0.

	0x0000	00000000	00000000	Q=0	S=0	L=0

The conditions under which each of these quality states will be reported are as follows:

1. Good

- The Communications link has been verified.
- The PLC understood our Poll request and returned a valid response packet.
- If a write occurred, there were no errors during the write process.
- There were no conversion problems with the data contained in the response packet.

Example The value 0x0000A is returned due to a poll of a register containing 10 (decimal).

2. Clamped High

- The Communications link has been verified.
- The PLC understood our Poll request and returned a valid response packet.
- The register was read or written without error.
- It was necessary to clamp its intended value to a limit because the value was larger than the maximum allowed.
- In the case of a string, the string is truncated.

Example An unsigned 16 bit integer is clamped to 65535.

3. Clamped Low

- The Communications link has been verified.
- The PLC understood our Poll request and returned a valid response packet.
- The register was read or written without error.
- It was necessary to clamp its intended value to a limit because the value was smaller than the minimum allowed.

Example An unsigned 16 bit integer is clamped to 0.

4. Cannot Convert

- The Communications link has been verified.
- The PLC understood our Poll request and returned a valid response packet.
- The data from the PLC could not be converted into the desired format.
- Possibilities for cannot convert include, but are not limited to:
  - The Server may return a constant in place of the data or return quality information alone.
  - The data is not usable.
  - It is not known whether the value is too large or too small.
  - The data returned from the PLC is of the incorrect data type.
  - A Floating Point number is returned, but is not value (for example: Not A Number).

Example The value of 0x000A is returned from a BCD register in a PLC.

5. Cannot Access Point

- The Communications link has been verified.
- The PLC understood our Poll request and returned a valid response packet.
- The PLC reported that it could not access the requested point.
- Possibilities for lack of accessibility include, but are not limited to:
  - Item does not exist in PLC memory.
  - Item is not currently available (locked in some way due to resource contention).
  - Item is not of the correct format / data type.
  - A write attempt was made, but item is read-only.
  - In most cases, a group of items will be affected when one item is invalid – this is due to the block-polling scheme used by the Servers. For example, if one item in a block of 10 is invalid, then entire block is marked invalid by the PLC. The Server will report invalid quality for all items in the block.
  - The data is unusable.

Example Attempting to read R40001 but R40001 is not defined in the PLC's memory map.

6. Communications Failed

- Any combination of the following:
  - Data communications are down.
  - The Topic is in slow poll (or equivalent) mode.
  - There have been no link validating messages.
  - Lack of resources in the Server. For example, a TSR (or driver) cannot allocate memory.
  - Lack of resources in the communications link.
  - The communications link is off-line.
  - All communications channels are in use.
  - The network is unable to route the message to the PLC.

Example Attempting to read data from a PLC which has been powered off.

See Also **.QualityLimit, .QualityStatus, .QualitySubstatus**

.QualityLimit tagname

Integer used to display the quality limit of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*`.QualityLimit`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Data Type Integer (read-only)

Valid Values (LLL)

0	Not Limited
1	Low Limited
2	High Limited
3	Constant

See Also **.Quality**

.QualityLimitString tagname

Used to display the quality limit string of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*`.QualityLimitString`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Data Type Message (read-only)

See Also **.QualityLimit, .Quality**

.QualityStatus tagname

Integer used to display the quality status of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**QualityStatus**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Remarks The Substatus Bit-Field (SSSS) is dependent on the value of the Quality Field (QQSSSSLL).

Data Type Integer (read-only)

Valid Values (SSSS)

0	Bad
1	Uncertain
3	Good

See Also **.QualitySubStatus, .Quality**

.QualityStatusString tagname

Used to display the quality status string of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**QualityStatusString**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Data Type Message (read-only)

See Also **.QualityStatus, .QualitySubStatus, .Quality**

.QualitySubstatus tagname

Integer used to display the quality substatus of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*`.QualitySubstatus`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname 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	Com 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=2):

0	Non-specific
6	Local Override

See Also **.QualityStatus, .Quality**

.QualitySubstatusString tagname

Used to display the quality substatus string of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*`.QualitySubstatusString`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog, or Message tagname type.

Data Type Message (read-only)

See Also **.QualityStatus, .QualitySubstatus, .Quality**

.RawValue tagname

The actual value received from an I/O Server by WindowViewer as a client. The raw value field allows the user to access the value of an I/O tagname before InTouch applies scaling.

Usage

`tagname.RawValue`

Parameters	Description
Tagname	Any I/O Discrete, Indirect Discrete, I/O Integer, Memory Real, Indirect Analog, I/O Message and Indirect Message tagname type.

Remarks

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

Data Type

Integer (read-only)

Valid Values

Where discrete or analog values can be specified.

Example

The following could be used to determine if a tagname is out of normal operating range.

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

See Also

.MinRaw, .MaxRaw, .MinEU, .MaxEU

.Reference tagname

Allows the operator to dynamically change the Access Name and/or Item Name during runtime.

Usage *TagName*`.Reference`

Parameters	Description
Tagname	Any I/O Discrete, Indirect Discrete, I/O Integer, I/O Real, Indirect Analog, I/O Message and Indirect Message tagname type.

Remarks This dot field provides an easy way to dynamically change the Access Name and /or Item of a tagname.

Data Type Message (read/write)

Valid Values Any string that contains an Access Name and/or Item.

Example The following statement sets the item field of a I/O Integer to an item name generated by a text function, and is determined by the value of a memory integer. This would be used within a condition script, based on **.ReferenceComplete** to cycle through a sequence of tags:

```
MyTag.Reference="R" + Text( MyIndex, "#" );
{ If MyIndex=40001,the resulting ITEM would be:  R40001 }
```

.ReferenceComplete tagname

Returns a confirmation when the **.Reference** item tagname changed, receives the item requested and the updated is reflected in the **.Value** field for that tagname.

Usage *TagName*`.ReferenceComplete;`

Parameters	Description
Tagname	Any I/O Discrete, Indirect Discrete, I/O Integer, I/O Real, Indirect Analog, I/O Message and Indirect Message tagname type.

Remarks This dot-field is very useful when using the **.Reference** field. **.ReferenceComplete** indicates that the item and/or access name has been changed, and more importantly, that a new value has been received from the new data source. Even if the new data source's value is identical to the previous data source, when the first update occurs from the new data source, this dot-field will be set equal to 1.

During the actual update process, from the time **.Reference** is modified until the time the first update from the new data source arrives, this dot-field's value will be set equal to 0 by the system.

Data Type I/O Discrete (read-only)

.ROCPct alarms

Monitors and/or controls the rate of change for alarm checking.

Usage

Tagname.**ROCPct**

Parameters	Description
Tagname	Any Integer, Real and Indirect Analog tagname type.

Remarks

This dot field is expressed in percentage. The dot field corresponds directly to the same field configured within the alarm section of the tagname dictionary. The user may display this dot field or use it in a script. Additionally, the value can be changed during runtime to accommodate changes to the process being monitored.

Data Type

Integer (read/write)

Valid Values

0 to 100%

Example

The following statement sets the Rate of Change Percent property of the tag named **MyTag** to a value of 25%:

```
MyTag.ROCPct=25;
```

See Also

.ROCStatus

.ROCStatus alarms

Determines whether a Rate-of-Change alarm exists for the specified tagname.

Usage *Tagname*.**ROCStatus**

Parameters	Description
Tagname	Any Integer, Real and Indirect Analog tagname type.

Remarks This read-only dot field is normally equal to 0. When a rate-of-change alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists.

Data Type Discrete (read-only)

Valid Values 0 = Specified Alarm Condition is not Present

1 = Specified Alarm Condition is Present

Example The body of the following IF-THEN statement will be processed only when the .ROCStatus ("Rate Of Change Alarm") of the tagname named **MyTag** is equal to a value of 1. In other words, when MyTag goes into Rate-Of-Change-Alarm, the body of the IF-THEN will execute.

```
IF (MyTag.ROCStatus == 1) THEN
OperatorMessage="MyTag has gone into a Rate-Of-Change-
Alarm";

ENDIF;
```

Remarks This dot field is often used in conjunction with the **.Alarm** and **.Ack** fields to determine the exact nature of the alarm status of a particular tagname within the system.

See Also **.ROCPct**

.ScooterLockLeft historical

Setting this dot field to a value equal to 1 (TRUE) will not allow the Right Scooter from moving to the Left of (overtake) the Left Scooter's position.

Usage

Tagname`.ScooterLockLeft`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

In general, it is beneficial to keep the user from being able to move the Right Scooter farther left than the Left Scooter's current position. When the left scooter is not locked, the right scooter will force the left scooter position to be equal to the right scooter position whenever the right scooter overtakes the left scooter.

Data Type

Discrete (read/write)

Valid Values

0 = FALSE = Right Scooter *can* overtake the Left Scooter's position

1 = TRUE = Right Scooter *cannot* overtake the Left Scooter's position

Example

This statement, once processed, will setup the Right Scooter associated with the Historical Trend Tag named "MyHistTrendTag", such that it will not be able to move to the left of the Left Scooter's current position.

```
MyHistTrendTag.ScooterLockLeft=1;
```

See Also

.ScooterPosRight, .ScooterPosLeft, .ScooterLockRight

.ScooterLockRight historical

Setting this dot field to a value equal to 1 (TRUE) will not allow the Left Scooter from moving to the Right of (overtake) the Right Scooter's position.

Usage

Tagname`.ScooterLockRight`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

In general, it is beneficial to keep the user from being able to move the Left Scooter farther right than the Right Scooter's current position. When the right scooter is not locked, the left scooter will force the right scooter position to be equal to the left scooter position whenever the left scooter overtakes the right scooter.

Data Type

Discrete (read/write)

Valid Values

0 = FALSE = Left Scooter *can* overtake the Right Scooter's position

1 = TRUE = Left Scooter *cannot* overtake the Right Scooter's position

Example

This statement, once processed, will setup the Left Scooter, associated with the Historical Trend Tag named "MyHistTrendTag", such that it will not be able to move to the right of the Right Scooter's current position.

```
MyHistTrendTag.ScooterLockRight=1;
```

See Also

.ScooterPosRight, .ScooterPosLeft, .ScooterLockLeft

.ScooterPosLeft historical

Monitors and/or controls the position of the Left Scooter.

Usage

Tagname`.ScooterPosLeft`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

This read/write dot field dynamically controls the position of the Left Scooter. The user may use this field in a script function to retrieve the current position of the Left Scooter, or the user may assign a value to this field to adjust the position of the Left Scooter to another location on the trend.

This field is most often used in conjunction with the **HTGetValue()** function collection. These functions must be told which Historical Trend is being queried, as well as the current position of the trend's scooters. See the following example.

Data Type

Real (read/write)

Valid Values

0.0 to 1.0; where 0 is the extreme left hand side of the Historical Trend chart, and 1.0 is the extreme right hand side of the Historical Trend chart.

Example

The following statement assigns a new location to the Left Scooter. The Left Scooter will be moved to a location 34% of the chart's total length away from the extreme left hand side of the Historical Trend chart currently associated with the Historical Trend tag named "MyHistTrendTag."

```
MyHistTrendTag.ScooterPosLeft=.34;
```

In the following statement, the script function **HTGetValueAtScooter()** is used to retrieve the value of Pen1 at the Left Scooter's current position. Since a change to any variable within a function's parameter list causes the function to be re-evaluated, each time the position of the Left Scooter changes, this statement will be re-evaluated.

```
MyRealTag=HTGetValueAtScooter( MyHistTrendTag,
   MyHistTrendTag.UpdateCount,1,
   MyHistTrendTag.ScooterPosLeft,1,"PenValue");
```

See Also

.ScooterPosRight, .ScooterLockLeft, ScooterLockRight

.ScooterPosRight historical

Monitors and/or controls the position of the Right Scooter.

Usage *Tagname*`.ScooterPosRight`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks This read/write dot field dynamically controls the position of the Right Scooter. The user may use this field in a script function to retrieve the current position of the Right Scooter, or the user may assign a value to this field to adjust the position of the Right Scooter to another location on the trend.

This field is most often used in conjunction with the **HTGetValue()** function collection. These functions must be told which Historical Trend is being queried, as well as the current position of the trend's scooters. See the following example.

Data Type Real (read/write)

Valid Values 0.0 to 1.0; where 0 is the extreme right hand side of the Historical Trend chart, and 1.0 is the extreme left hand side of the Historical Trend chart.

Examples The following statement assigns a new location to the Right Scooter. The Right Scooter will be moved to a location 34% of the chart's total length away from the extreme right hand side of the Historical Trend chart which is currently associated with the Historical Trend tag named "MyHistTrendTag."

```
MyHistTrendTag.ScooterPosRight=.34;
```

This statement uses the script function **HTGetValueAtScooter()** to retrieve the value of Pen1 at the Right Scooter's current position. Since 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 will be re-evaluated.

```
MyRealTag=HTGetValueAtScooter( MyHistTrendTag,
    MyHistTrendTag.UpdateCount,2,
    MyHistTrendTag.ScooterPosRight,1,"PenValue");
```

See Also **.ScooterPosLeft, .ScooterLockLeft, ScooterLockRight**

.SPCStatus SPC

Determines whether an SPC alarm exists for the specified tagname.

Note This alarm dot field is only applicable if the SPC program is installed.

Usage *Tagname*.**SPCStatus**

Parameters	Description
Tagname	Any Integer, Real and Indirect Analog tagname type.

Data Type Discrete (read-only)

Valid Values 0 = SPC Alarm Condition is not Present

1 = SPC Alarm Condition is Present

Remarks This read-only dot field is normally equal to 0. When an SPC alarm condition exists for the specified tagname, this dot field is set to a value of 1 by the system. The dot field will remain equal to 1 until the alarm condition no longer exists.

Example The body of the following IF-THEN statement will be processed only when the **.SPCStatus** ("Statistical Process Control Alarm") of the tag named **MyTag** is equal to a value of 1. In other words, when MyTag goes into SPC-Alarm, the body of the IF-THEN will execute.

```
IF (MyTag.SPCStatus == 1) THEN
OperatorMessage="MyTag has gone into an SPC-Alarm";
ENDIF;
```

.TagID tagname

Used in conjunction with the Historical Trend .**Pen1**-.**Pen8** TagID tagnames to monitor and/or control the tagname being trended by a pen.

Usage *Tagname*.**TagID**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Discrete and Indirect Analog tagname type.

Remarks .TagID provides the handle of a tagname and is used mainly in the context of assigning tags to pens of a Historical Trend.

Data Type TagID (read-only)

Example

```
MyHistTrendTag.Pen6=SomeAnalogTag.TagID;
```

See Also **.Pen1-.Pen8**

.TimeDate tagname

Integer Tagname.field used to display the whole number of days which have passed since an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname.*`TimeDate`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

.TimeDateString tagname

String that display the date in the same format set in the WIN.INI file.

Usage *Tagname.*`TimeDateString`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Message (read-only)

.TimeDateTime tagname

Real Tagname.field used to display the fractional number of days which have passed since an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname.*`TimeDateTime`

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Message (read-only)

.TimeDay tagname

Integer Tagname.field used to display the day an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeDay**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 1 - 31.

.TimeHour tagname

Integer Tagname.field used to display the hour of the day that an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeHour**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 0 - 23.

.TimeMinute tagname

Integer Tagname.field used to display the minute that an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeMinute**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 0 - 59.

.TimeMonth tagname

Integer Tagname.field used to display the month that an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeMonth**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 1 - 12.

.TimeMsec tagname

Integer Tagname.field used to display the time in milliseconds that an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeMsec**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 0 - 999.

.TimeSecond tagname

Integer Tagname.field used to display the time in seconds that an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeSecond**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 0 - 59.

.TimeTime tagname

Integer Tagname.field used to display the time in milliseconds (since midnight) that an I/O value was provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeTime**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values Value may be 0 - 59.

.TimeTimeString tagname

Message Tagname.field used to display the time and day of an I/O value provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeTimeString**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Message (read-only)

.TimeYear tagname

Integer Tagname.field used to display the year is four digits that an I/O value was provided by an I/O Server when the I/O connection is valid.

Usage *Tagname*.**TimeTime**

Parameters	Description
Tagname	Any Discrete, Integer, Real, Indirect Analog and Message tagname type.

Data Type Integer (read-only)

Valid Values 1990

.Unack alarms

Controls the alarm unacknowledgment status of local alarm(s).

Usage

`Tagname.Unack=1`

Parameters	Description
Tagname	Any Discrete, Integer, Real Indirect and Group variable tagname type.

Remarks

Set this dot field to a value of 1 to unacknowledge any outstanding alarm(s) the specified tagname/group may have associated with it. When the specified tagname is of type "Group Var" or is an Alarm Group, all unacknowledged alarms associated with the tags within the specified group are acknowledged. When the specified tagname is of any type other than Group Var, only the acknowledged alarm associated with that tagname is unacknowledged. Setting this dot field to a value other than 1 has no meaning, and the results are undefined.

Data Type

Discrete (read/write)

Valid Values

1

Examples

The following statement unacknowledges an alarm associated with a tag named "Tag1."

```
Tag1.Unack=1;
```

This statement would be used to unacknowledge all acknowledged alarms within the alarm group named "PumpStation."

```
PumpStation.Unack = 1;
```

This statement is similar to the statement above, except it is used to unacknowledge any acknowledged alarms within the alarm group currently associated with the GroupVar tag named "StationAlarms."

Note **.Unack** has an inverse .field called **.Ack**. When an acknowledged alarm occurs, **.Ack** will be set to **1**.

See Also

.Ack and .Alarm

.UpdateCount historical

Incremented each time an update has occurred for the associated Trend.

Usage

Tagname.**UpdateCount**

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

This dot field is linked directly to the historical retrieval sub-system. Whenever new data has been requested of the Historical Trend associated with the specified Historical Trend Tag, the historical retrieval sub-system must go out to disk and retrieve the specified data. Once the retrieval process has been completed, this dot field is incremented by a value of 1. **.UpdateCount** is used in many of the function calls related to Historical Data Trends as a way to trigger the recalculation of the function.

Data Type

Integer (read-only)

Valid Values

Any positive integer

Example

The following statement uses the script function **HTGetValueAtScooter()** to retrieve the value of Pen1 at the Right Scooter's current position. Since a change to any variable within a function's parameter list causes the function to be re-evaluated, each time a retrieval (update) has been completed, and the value of **.UpdateCount** is incremented, this statement will be re-evaluated.

```
MyRealTag=HTGetValueAtScooter( MyHistTrendTag,
   MyHistTrendTag.UpdateCount,2,
   MyHistTrendTag.ScooterPosRight,1,"PenValue");
```

See Also

.UpdateInProgress, **.UpdateTrend**

.UpdateInProgress historical

Equal to 1 if a Historical Retrieval is in progress; otherwise equal to 0.

Usage *Tagname*.**UpdateInProgress**

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks This dot field is linked directly to the historical retrieval sub-system. Whenever new data has been requested of the Historical Trend associated with the specified Historical Trend Tag, the historical retrieval sub-system must go out to the disk and retrieve the specified data. During the retrieval process this dot field is set equal to 1. Once the process has completed, .**UpdateInProgress** is reset to 0. **.UpdateInProgress** is used in many of the function calls related to Historical Data Trends.

Most Historical Trend screens have a mechanism by which an operator can "scroll" through data on the Trend. As the operator manipulates the controls on the screen, the Historical Subsystem must make sure the data on the screen is current. If the operator scrolls the trend to a section in time that is not currently displayed or sitting in memory, the subsystem will go out to disk and retrieve the requested data. Since that process could take some time, the system provides a way for the designer of the Historical Trend screen to make the operator aware that the requested data is currently being retrieved. Without that kind of feedback, the operator may not be aware that the system is performing the requested task.

Data Type Discrete (read-only)

Value Values 0 = No update in progress

1 = Update in progress

Examples The following statement is commonly used as the expression in a "visibility" link on a text object near or on the "scroll" buttons on the Historical Trend. When the historical subsystem is retrieving the requested data, this expression evaluates to a value of 1. Otherwise, if the trend is up to date, this expression evaluates to a value of 0.

```
MyHistTrendTag.UpdateInProgress
```

See Also **.UpdateCount, .UpdateTrend**

.UpdateTrend historical

Causes a Historical Trend chart to update using all current values.

Usage

Tagname`.UpdateTrend`

Parameters	Description
Tagname	Any Hist Trend tagname type.

Remarks

Historical Trends do not automatically update themselves. A change must be made to either the Chart Start or Chart Length in order for the chart to update and display the current values for the specified tagnames. Using this dot field in an action pushbutton script will allow the operator to update the chart whenever necessary during runtime.

It can also be used within a script if other dot fields associated with the Hist Trend are going to be changed. This will ensure an up-to-date display on the Historical Trend Chart.

Setting this dot field to a value other than 1 has no meaning, and the results are undefined.

Data Type

Discrete (write only)

Valid Values

1

Example

The following statement causes the Historical Trend associated with the Hist Trend tag named "MyHistTrendTag" to update with the current values of all parameters:

```
MyHistTrendTag.UpdateTrend=1;
```

.Value tagname

Contains the value of the specified tagname. This is also the default dot field for every InTouch tagname within the system. If no other dot field is specified, this dot field is assumed.

Usage

Tagname`.Value`

Parameters	Description
Tagname	Any tagname except Hist Trend tagname types.

Remarks

This is the default dot field for every InTouch tagname within the system. If no other dot field is specified, this dot field is assumed. Rarely would you need to use this dot field, but in some instances, it makes a calculation or parameter usage clearer, and will therefore, be useful as a documentation tool.

Data Type

Depends on (is the same as) the specified tagname's type (read/write).

Example

The following statement sets the value of the Memory Integer Tag named **MyTag** equal to 100.

```
Tagname.Value=100;
```

Which is functionally identical to:

```
Tagname=100;
```

.AlarmGroup distributed alarms

Contains the current query list of a distributed alarm display.

Usage

```
[ErrorNumber=]GetPropertyM("ObjectName.AlarmGroup",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	A message tagname that will hold the property value when the function is processed.

Remarks This read-only dot field contains the current alarm query used by the named distributed alarm display. This query may be a list of alarms groups or direct alarm provider references.

Data Type Message (read-only)

Example This statement returns the current alarm query used by display object "AlmObj_1" to message tagname "CurrentQuery":

```
GetPropertyM("AlmObj_1.AlarmGroup",CurrentQuery)
```

See Also **GetPropertyM()**

.NextPage distributed alarms

Scrolls the alarm display one page (one screen full of alarms) down when this property transitions from 1 to 0.

Usage

```
[ErrorNumber=]GetPropertyD("ObjectName.NextPage",Tagname);
[ErrorNumber=]SetPropertyD("ObjectName.NextPage",Value);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj1*.
Tagname	The name of a discrete tagname that will hold the property value when the function is processed used as "Tagname" or Tagname.Name.
Value	A discrete value. Or, a discrete InTouch tagname that holds the value to be written when the function is processed.

Remarks Whenever this value transitions from 1 to 0, the alarm display will display the next page. Once the next page is displayed, the variable will automatically set to 1, unless the top of the list has been reached. In this case, the value remains 0.

Data Type Discrete (read/write)

See Also **GetPropertyD(), SetPropertyD(), .PrevPage**

.NumAlarms distributed alarms

Contains the number of alarms within an alarm object.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.NumAlarms",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer type tagname that will hold the property value when the function is processed.

Remarks This read-only dot field contains the current number of alarms registered in a named distributed alarm display. This includes not only those alarms displayed, but all alarms registered.

Data Type Integer (read-only)

Example The following statement returns the current number of alarms used by display object "AlmObj_1" to integer tagname "AlarmCount":

```
GetPropertyI("AlmObj_1.NumAlarms",AlarmCount);
```

See Also **GetPropertyI()**

.PageNum distributed alarms

Contains the current page numbers displayed in the alarm object.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.PageNum",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer type tagname that will hold the property value when the function is processed.

Remarks This read-only dot field contains the number of the currently displayed page in a named distributed alarm display.

Data Type Integer (read-only)

Example The following statement returns the current page number of display object "AlmObj_1" to integer tagname "AlarmPage":

```
GetPropertyI("AlmObj_1.PageNum",AlarmPage);
```

See Also **GetPropertyI()**

.PrevPage distributed alarms

Scrolls the alarm display one page (one screen full of alarms) up when this property transitions from 1 to 0.

Usage

```
[ErrorNumber=]GetPropertyD("ObjectName.PrevPage",Tagname);
[ErrorNumber=]SetPropertyD("ObjectName.PrevPage",Value);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	A tagname (of the same type to be returned) that will hold the property value when the function is processed.
Value	A discrete value. Or, a discrete InTouch tagname that holds the value to be written when the function is processed.

Remarks

Whenever this value transitions from 1 to 0, the alarm display will display the previous page. Once the previous page is displayed, the variable will automatically set to 1, unless the top of the list has been reached. In this case, the value remains 0.

Data Type

Discrete (read/write)

See Also

GetPropertyD(), SetPropertyD(), .NextPage

.PriFrom distributed alarms

Contains the current query priority low filter value.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.PriFrom",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the minimum priority value used in a named distributed alarm display to query for alarms.

Data Type

Integer (read-only)

Example

The following statement returns the minimum priority value of the query used by display object "AlmObj_1" to integer tagname "MinPri":

```
GetPropertyI("AlmObj_1.PriFrom",MinPri);
```

See Also

GetPropertyI(), .PriTo

.PriTo distributed alarms

Contains the current query priority high filter value.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.PriTo",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks This read-only dot field contains the maximum priority value used in a named distributed alarm display to query for alarms.

Data Type Integer (read-only)

Example The following statement returns the maximum priority value of the query used by display object "AlmObj_1" to integer tagname "MaxPri":

```
GetPropertyI("AlmObj_1.PriTo",MaxPri);
```

See Also **GetPropertyI(), .PriFrom**

.ProviderReq distributed alarms

Contains the number of alarm providers required by the current query.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.ProviderReq",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks This read-only dot field contains the number of alarm providers required by the current query used by a named distributed alarm display.

Data Type Integer (read-only)

Example The following statement returns the number of alarm providers required by the current query used by display object "AlmObj_1". This value is written to integer tagname "TotalProv":

```
GetPropertyI("AlmObj_1.ProviderReq",TotalProv);
```

See Also **GetPropertyI(), .ProviderRet**

.ProviderRet distributed alarms

Contains the number of alarm providers that have successfully returned their query results.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.ProviderRet",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the number of alarm providers returned by the current query used by a named distributed alarm display.

Data Type

Integer (read-only)

Example

The following statement returns the number of alarm providers that have successfully returned their alarms to display object "AlmObj_1". This value is written to integer tagname "RetProv":

```
GetPropertyI("AlmObj_1.ProviderRet",RetProv);
```

See Also

GetPropertyI(), .ProviderReq

.QueryState distributed alarms

Represents the current alarm state query filter.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.QueryState",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the current query filter used by a named distributed alarm display.

Data Type

Integer (read-only)

Valid Values

0 = All

1= Unack

2 = Ack

Example

The following statement returns the current query filter of display object "AlmObj_1" to integer tagname "AlmQueryState":

```
GetPropertyI("AlmObj_1.QueryState",AlmQueryState);
```

See Also

GetPropertyI()

.QueryType distributed alarms

Represents the current alarm query type.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.QueryType",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the current query type used by a named distributed alarm display.

Data Type

Integer (read-only)

Valid Values

1 = History

2 = Summary

Example

The following statement returns the current query type of display object "AlmObj_1" to integer tagname "AlmQueryType":

```
GetPropertyI("AlmObj_1.QueryType",AlmQueryType);
```

See Also

GetPropertyI()

.Successful distributed alarms

Represents whether the current query was successful.

Usage

```
[ErrorNumber=]GetPropertyD("ObjectName.Successful",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	A discrete tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the state of the last query used by a named distributed alarm display.

Data Type

Discrete (read-only)

Valid Values

0 = Error in query

1 = Successful query

Example

The following statement returns the state of the last query of display object "AlmObj_1" to discrete tagname "AlmFlag":

```
GetPropertyD("AlmObj_1.Successful",AlmFlag);
```

See Also

GetPropertyD()

.TotalPages distributed alarms

Contains the total number of pages (one screen full of alarms) in the alarm object.

Usage

```
[ErrorNumber=]GetPropertyI("ObjectName.TotalPages",Tagname);
```

Parameters	Description
ObjectName	Name of the alarm object, e.g., *AlmObj_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.

Remarks

This read-only dot field contains the total number of alarm pages contained in a named distributed alarm display.

Data Type

Integer (read-only)

Example

The following statement returns the total number of alarm pages contained in display object "AlmObj_1" to integer tagname "AlmTotalPage":

```
GetPropertyI("AlmObj_1.TotalPages",AlmTotalPages);
```

See Also

GetPropertyI()

.Caption windows control

Determines the "message" to be displayed with the check box.

Usage

```
[ErrorNumber=]GetPropertyM ("ControlName.Caption", Tagname );
[ErrorNumber=]SetPropertyM ("ControlName.Caption", "Message");
```

Parameters	Description
ControlName	Name of the windows control, e.g., *ChkBox_4*.
Tagname	A message tagname that will hold the property requested.
"Message"	A message string surrounded in quotes.

Remarks This property is read/write during both development and runtime.

Data Type Message (read/write)

Applies To Check boxes.

Example This statement sets the Caption of the Check box object "CheckBox_1" equal to "Blue Paint Option."

```
SetPropertyM( "CheckBox_1.Caption","Blue Paint Option" );
```

See Also **GetPropertyM(), SetPropertyM()**

.Enabled windows control

Determines whether the control object can respond to user-generated events.

Usage

```
[ErrorNumber=] GetPropertyD("ControlName.Enabled", Tagname);
[ErrorNumber=] SetPropertyD("ControlName.Enabled", Discrete);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *ChkBox_4*.
Tagname	A discrete tagname that will hold the property requested.
Discrete	A discrete value: 0 = Control is disabled 1 = Control is enabled -Or- A discrete tagname that holds the value to be written when the function is processed.

Remarks This property is read/write during both development and runtime.

Data Type Discrete (read/write)

Applies To Text boxes, List boxes, Combo boxes, Check boxes and Radio buttons.

Example The following statement disables the List box object called: "ListBox_1."

```
SetPropertyD( "ListBox_1.Enabled",0 );
```

See Also **GetPropertyD(), SetPropertyD()**

.ListCount windows control

Determines the number of items in the List box or Combo box.

Usage

```
[ErrorNumber=]GetPropertyI("ControlName.ListCount",Tagname);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *ListBox_4*.
Tagname	A valid tagname that contains the integer count of the items in the list.

Remarks This property is available only at runtime.

Data Type Integer (read-only)

Applies To List boxes and Combo boxes.

Example The following statement retrieves the number of items in the List box named "ListBox_1" and places that value into the memory integer tagname named "MyListBoxCount."

```
GetPropertyI( "ListBox_1.ListCount",MyListBoxCount );
```

See Also **GetPropertyI()**

.ListIndex windows control

Determines the corresponding index (*Tagname* or *Number*) of the currently selected item in the list. Index is a number that defines a specific item in a list. When using a List box, an index of -1 indicates that no item is currently selected. When using a Combo box, an index of -1 indicates that the user has entered new text into the text entry field of the control.

Usage

```
[ErrorNumber=]GetPropertyI("ControlName.ListIndex",Tagname);
[ErrorNumber=]SetPropertyI("ControlName.ListIndex",Number);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *ListBox_4*.
Tagname	A valid tagname that contains the integer count of the items in the list.
Number	The index number that defines a specific item in the list.

Remarks

This property is available only at runtime.

Data Type Integer (read/write)

Applies To List boxes and Combo boxes.

Example

This statement retrieves the index of the currently selected item in the List box named "ListBox_1" and places that value into the memory integer tag named "MyListBoxIndex."

```
GetPropertyI( "ListBox_1.ListIndex",MyListBoxIndex );
```

See Also

GetPropertyI(), SetPropertyI()

.NewIndex windows control

Returns the corresponding integer index (*Tagname*) of the last item added to the List box or Combo box via the **wcAddItem()** or **wcInsertItem()**.

Usage

```
[ErrorNumber=]GetPropertyI("ControlName.NewIndex",Tagname);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *ListBox_4*.
Tagname	A tagname containing the integer index of the last item added to the List or Combo box. For empty lists, a value of -1 is returned.

Remarks This property is only available at runtime.

Data Type Integer (read-only)

Applies To List boxes and Combo boxes.

Example The following statement retrieves the index of the most recently added item in the List box named "ListBox_1" and places that value into the memory integer tag named "NewItemIndex."

```
GetPropertyI("ListBox_1.NewIndex",NewItemIndex);
```

See Also **GetPropertyI(), wcAddItem(), wcInsertItem()**

.ReadOnly windows control

Determines whether the contents of the Text box are read-only or read-write.

Usage

```
[ErrorNumber=]GetPropertyD("ControlName.ReadOnly", Tagname);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *Textbox_1*.
Tagname	A discrete tagname that will hold the property value when the function is processed.

Remarks This property is available in both development and runtime.

Data Type Discrete (read-only)

Valid Values 0 = Contents of Text box is read/write

1= Contents of Text box is read-only

Applies To Text boxes.

Example The following statement retrieves the read-only status of the Text box named "TextBox_1":

```
GetPropertyD( "TextBox_1.ReadOnly",A_Tagname );
```

See Also **GetPropertyD(), SetPropertyD()**

.TopIndex windows control

Determines the corresponding integer index of the top-most item in a List box.

Usage

```
[ErrorNumber=]GetPropertyI("ControlName.TopIndex", Tagname);
[ErrorNumber=]SetPropertyI("ControlName.TopIndex", Number);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *Listbox_1*.
Tagname	An integer tagname that will hold the property value when the function is processed.
Number	The index number that defines the top-most item in the List box.

Remarks This property is available only at runtime.

Data Type Integer (read/write)

Applies To List boxes.

Example The following statement sets the TopIndex of the List box object "ListBox_1" to a value of 14:

```
SetPropertyI( "ListBox_1.TopIndex",14 );
```

See Also **GetPropertyI(), SetPropertyI()**

.Value windows control

The default property for all InTouch windows control wizards. Changes made to this property are synchronized in the InTouch tagname and the windows control wizards.

Usage

```
[ErrorNumber=]GetPropertyM("ControlName[.Value]", Tagname);
[ErrorNumber=]SetPropertyM("ControlName[.Value]", Value);
[ErrorNumber=]GetPropertyI("ControlName[.Value]", Tagname);
[ErrorNumber=]SetPropertyI("ControlName[.Value]", Value);
[ErrorNumber=]GetPropertyD("ControlName[.Value]", Tagname);
[ErrorNumber=]SetPropertyD("ControlName[.Value]", Value);
```

Note The initial value of tagnames assigned to either a List box or Combo box cannot be used to initialize the value of the List box or Combo box.

Parameters	Description
ControlName	Name of the windows control, e.g., *ChkBox_4*.
[**.Value**]	This property is optional. If not specified, the function will always assume the **.Value** property is being used.
Tagname	A tagname (of the same type to be returned) that will hold the property value when the function is processed.
Value	The actual value to be written or a valid InTouch tagname (of the same type as the property to be written to) that holds the property value to be written when the function is processed.

Remarks This property is read/write during development and runtime. If the **.Value** is accessed by associating a tagname to either a List box or a Combo box, it is read-only. If the .**Value** is assigned to a Check box, Radio button, or Text box it is read/write. The value specified at development serves as the default for runtime.

Data Type Message (read/write) for Text boxes, List boxes and Combo boxes.

Integer (read/write) for Radio buttons

Discrete (read/write) for Check boxes.

Applies To Text boxes, List boxes, Combo boxes, Check boxes and Radio buttons.

Example The following statement sets the Value of the Radio Button object "RadioButton_1" to a value of 4:

```
SetPropertyI( "RadioButton_1.Value",4 );
```

See Also **GetPropertyM(), SetPropertyM(), GetPropertyI(), SetPropertyI(), GetPropertyD(), SetPropertyD()**

.Visible windows control

Determines whether the windows control is visible in the window.

Usage

```
[ErrorNumber=]GetPropertyD("ControlName.Visible",Tagname);
[ErrorNumber=]SetPropertyD("ControlName.Visible",Number);
```

Parameters	Description
ControlName	Name of the windows control, e.g., *Listbox_1*.
Tagname	A tagname (of the same type to be returned) that will hold the property value when the function is processed.
Number	A discrete value. Or, a discrete InTouch tagname that holds the value to be written when the function is processed.

Remarks This property is read/write in both development and runtime.

Data Type Discrete (read/write)

Valid Values 0 = Control is invisible

1= Control is visible

Applies To Text boxes, List boxes, Combo boxes, Check boxes and Radio buttons.

Example The following statement creates a Text box named "TextBox_1" invisible:

```
SetPropertyD( "TextBox_1.Visible",0 );
```

See Also **GetPropertyD(), SetPropertyD()**

CHAPTER 3

InTouch QuickScript Functions

InTouch scripting is one of the most powerful features of an InTouch application. The InTouch QuickScript capabilities allow you to execute commands and logical operations based on specified criteria being met. For example, a key being pressed, a window being opened, a value changing, and so on.

QuickFunctions are scripts that you create that can be called from other scripts and animation link expressions. The reused code is stored in one script and in one location, thereby supporting update of all script instances with one edit session.

By using scripts, a wide variety of customized and automated system functions can be created.

Abs() math

Returns the absolute value (unsigned equivalent) of a specified number.

Syntax

```
Result=Abs(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks

The absolute value of *Number* is calculated and returned to *Result.*

Examples

```
Abs(14) will return 14
Abs(-7.5) will return 7.5
```

Ack() alarm

Acknowledges any unacknowledged InTouch alarm.

Syntax

```
Ack Tagname;
```

Parameter	Description
Tagname	Any InTouch tagname, Alarm Group, or Group Variable.

Remarks

This function can be applied to a tagname, Alarm Group or Group Variable. (A Group Variable is a tagname that has the name of an Alarm Group assigned to it.)

Examples

The following statements can be used on a pushbutton to acknowledge any unacknowledged alarm:

```
Ack $System; (All alarms)
Ack Tagname;
Ack GroupName;
Ack GroupVariable;
```

ActivateApp() system

Activates another currently running windows application.

Syntax

```
ActivateApp TaskName;
```

Parameter	Description
TaskName	The task this function will activate.

Remarks

TaskName is the exact text string, including spaces, that appears on the Task Bar or in the Task Manager (accessed by right clicking Task Bar in Windows NT and then click Task Manager or Ctrl+Alt+Delete).

Example

The following example checks to see if the cmd prompt is running. If it is running, it brings it to the foreground and gives it focus. Otherwise, it launches a cmd prompt and starts the edit.com DOS program inside the cmd prompt.

```
IF InfoAppActive( InfoAppTitle("cmd")) == 1 THEN
   ActiveApp InfoAppTitle("cmd");
ELSE
   StartApp "cmd /c edit";
ENDIF;
```

See Also

StartApp(), InfoAppTitle()

almAckAll() distributed alarms

Acknowledges all alarms in current query including those not currently displayed in the alarm display.

Syntax

```
[Result=]almAckAll(ObjectName,Comment);
```

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.
Comment	Alarm acknowledgment comment.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
MessageTag = “Acknowledge All by” + $Operator;
AlmAckAll(“AlmObj_1”, MessageTag);
```

almAckDisplay() distributed alarms

Acknowledges only those alarms currently visible in the alarm display object.

Syntax `[Result=]almAckDisplay(ObjectName,Comment);`

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.
Comment	Alarm acknowledgment comment.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
almAckDisplay("AlmObj_1", "Display Acknowledgement");
```

almAckRecent() distributed alarms

Acknowledges the most recent alarms that has occurred.

Syntax `[Result=]almAckRecent(ObjectName,Comment);`

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.
Comment	Alarm acknowledgment comment.

Remarks For a list of returned error numbers, see Appendix A.

Example

```
almAckRecent("AlmObj_1", $DateString);
```

almAckSelect() distributed alarms

Acknowledges only those alarms selected in the alarm display object.

Syntax

```
[Result=]almAckSelect(ObjectName,Comment);
```

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.
Comment	Alarm acknowledgment comment.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
IF ($Hour > 0 and $Hour < 8) THEN
   AckTag = “NightShift”;
ELSE
   AckTag = “Day Shift”;
ENDIF;
AlmAckSelect (“AlmObj_1”,AckTag);
```

almDefQuery() distributed alarms

Performs a query to update an alarm display object using the default properties.

Syntax

```
[Result=]almDefQuery(ObjectName);
```

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.

Remarks

The default query properties are specified during development.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
almDefQuery(“AlmObj_1”);
```

almMoveWindow() distributed alarms

Scrolls the alarm display object window.

Syntax

```
[Result=]almMoveWindow(ObjectName,Options,Repeat);
```

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.
Option	The type of alarm acknowledge to perform:

Type	Description
LineDn	One line down.
LineUp	One line up.
PageDn	One page down.
PageUp	One page up.
Top	To the top of the list.
Bottom	To the bottom of the list.
PageRt	One page to the right.
PageLf	One page to the left.
Right	To the end of the list (right side).
Left	To the beginning of the list (left side).

Repeat	The number of times this operation should be repeated.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
almMoveWindow("AlmObj_1", "Bottom", 0);
almMoveWindow("AlmObj_1", "LineDn", 3);
almMoveWindow("AlmObj_1", "PageUp", 0);
```

almQuery() distributed alarms

Performs a query to update an alarm display object.

Syntax

```
[Result=]almQuery(ObjectName,AlarmList,FromPri,
    ToPri,State,Type);
```

Parameter	Description
ObjectName	The name of the alarm object. For example, *AlmObj_1*.
AlarmList	Sets the Alarm Group/Group List to perform the query against, for example, "*MyGroup*" or message tagname.
FromPri	Starting priority of alarms to display. for example, *100* or integer tagname.
ToPri	Ending priority of alarms to display. For example, *900* or integer tagname.
State	Specifies type of alarms to display. For example, "UnAck" or message tagname. Valid states are All, UnAck or Ack.
Type	Specifies type of query for example, "Hist" (Historical alarms) or "Summ" (Summary alarms).

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

This statement retrieves all of the historical alarms specified in "MyGroup" between priority 500 and 600. The alarms will populate alarm display "Alarm1."

```
AlmQuery("AlmObj_1","MyGroup",500,600,"All","Hist");
```

almSelectAll() distributed alarms

Toggles the selection of all the alarms in an alarm display object.

Syntax

```
[Result=]almSelectAll(ObjectName);
```

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
If $AccessLevel > 8000 THEN
   almSelectAll(“AlmObj_1”);
   almAckSelect(“AlmObj_1”, “Ack Selected by a Manager”);
ENDIF;
```

almSelectItem() distributed alarms

Toggles the selection of the item that is highlighted in an alarm display object.

Syntax `[Result=]almSelectItem(ObjectName);`

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
almSelectItem("AlmObj_1");
```

almShowStats() distributed alarms

Displays the alarm display object statistics screen.

Syntax `[Result=]almShowStats(ObjectName);`

Parameter	Description
ObjectName	The name of the alarm object. for example, *AlmObj_1*.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Example

```
almSlowStats("AlmObj_1");
```

ArcCos() math

Given a number between -1 and 1 (inclusive), this function will return an angle between 0 and 180 degrees whose *cosine* is equal to that number.

Syntax `Result=ArcCos(Number);`

Parameter	Description
Number	Any number, real or integer tagname.

Remarks The absolute value of *Number* is calculated and returned to *Result*.

Example

```
ArcCos(1) will return 0
ArcCos(-1) will return 180
```

ArcSin() math

Given a number between -1 and 1 (inclusive), this function will return an angle between -90 and 90 degrees whose *sine* is equal to that number.

Syntax

```
Result=ArcSin(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks The absolute value of *Number* is calculated and returned to *Result*.

Example

```
ArcSin(1) will return 90
ArcSin(-1) will return -90
```

ArcTan() math

Given a number, this function will return an angle between -90 and 90 degrees whose *tangent* is equal to that number.

Syntax

```
Result=ArcTan(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks The absolute value of *Number* is calculated and returned to *Result*.

Example

```
ArcTan(1) will return 45
ArcTan(0) will return 0
```

ChangePassword() security

Displays the **Change Password** dialog box allowing the logged on operator to change his/her password.

Syntax

```
[Result=]ChangePassword();
```

Parameters	Description
[Result]	Returns one of the following integer values: 0=Cancel was pressed. 1=OK was pressed.

Remarks If using touch screen applications, there is an option to use the alphanumeric keyboard.

Example

```
Errmgs=ChangePassword();
```

This script, if placed on a button or called based on a condition or data change script, will open a dialog (with optional keyboard) prompting the user to enter the current password, the new password and verification of the new password.

Cos() math

Returns the *cosine* of an angle given in degrees.

Syntax

```
Result=Cos(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks The absolute value of *Number* is calculated and returned to *Result*.

Example

```
Cos(90) will return 0
Cos(0) will return 1
Wave = 50 * Cos(6 * $Minute);
```

DialogStringEntry() misc

Displays an alphanumeric keyboard on the screen, allowing the operator to change the current string value of a message tagname in the Tagname Dictionary.

Syntax

`[Result=]`**`DialogStringEntry`**`(`*`MessageTag_Text,UserPrompt_Text`*`);`

Parameters	Description
Result	Returns one of the following integer values: 0=Cancel was pressed. 1=OK was pressed. -1=Internal error. -2=Could not initiate. -3=Tagname not defined. -4=Tagname is not a message type. -5=Unable to write.
MessageTag_Text	Specifies the name of the message tagname to be modified. (This function requires a string for this parameter, therefore, you must identify the tagname by its name within quotes or use the **.Name** field without quotes. A message tagname used as a pointer is also acceptable. See the following examples.
UserPrompt_Text	Specifies the user message to display at the top of the keyboard.

Remarks

This function is used primarily in applications containing touch screens.

Examples

```
Errmsg=DialogStringEntry(MyMessageTag.Name,
"Enter a new string...");
```

```
Errmsg=DialogStringEntry("MyMessageTag","Enter a new string...");
```

Parameters could also act as "pointers" to other tags, allowing the function to be dynamic during runtime.

```
Errmsg=DialogStringEntry(MessageTagX, MessageDisplay);
```

Using the above example, input links can be assigned to each of the pointer tags, allowing the user to modify any or all of the function parameters before processing the function.

For example, the following script would bring up an alphanumeric keyboard, allowing modification of MyMessageTag while displaying the message "Enter a new string..." at the top of the keyboard:

```
MessageTagX="MyMessageTag"; {assign the string MyMessageTag
(which is actually the tagname to be modified) to the memory
message tag MessageTagX}
```

```
MessageDisplay="Enter a new string..."; {assign the new message
string to the memory message tag MessageDisplay}
```

```
Errmsg=DialogStringEntry(MessageTagX, MessageDisplay); {quotes
are not required because MessageTagX was defined as a message
type tag}
```

DialogValueEntry() misc

Displays the numeric keypad on the screen, allowing the user to change the current value of a discrete, integer or real tagname in the Tagname Dictionary.

Syntax

```
[Result=]DialogValueEntry(ValueTag_Text,
    LowLimit,HighLimit,UserPrompt_Text);
```

Parameters	Description
Result	Returns one of the following integer values: 0=Cancel was pressed. 1=OK was pressed. -1=Highlimit<=Lowlimit. -2=Could not initiate. -3=Tagname not defined. -4=Tagname is not a discrete, integer or real type. -5=Write failed.
ValueTag_Text	Specifies the name of the discrete, integer or real tagname to be modified. (This function requires a string for this parameter; therefore, you must identify the tagname by its name within quotes or use the **.Name** field without quotes or use a message tagname as a pointer. See the following examples.)
LowLimit	Specifies the minimum allowable value for the tagname. (This should be >= the tagname's definition for Min Value, Min Raw or Min EU, as applicable).
HighLimit	Specifies the maximum allowable value for the tagname. (This should be <= the tagname's definition for Max Value, Max Raw or Max EU, as applicable).
UserPrompt_Text	Specifies the user message to display at the top of the keypad.

Remarks

This function is used primarily in applications containing touch screens.

Examples

```
Errmsg=DialogValueEntry(MyIntegerTag.Name, MyIntegerTag.MinEU,
MyIntegerTag.MaxEU, "Enter a  new value...");

Errmsg=DialogValueEntry("MyIntegerTag", -100, 100, "Enter a new
value...");
```

Parameters could also act as "pointers" to other tags, allowing the function to be dynamic during runtime.

```
Errmsg=DialogValueEntry(TagnameX, Min, Max, MessageDisplay);
```

Using the above example, input links can be assigned to each of the pointer tags, allowing the user to modify any or all of the function parameters before processing the function.

For example, the following script would bring up the numeric keypad, allowing modification of *MyIntegerTag* using a Min and Max limit of -100 and 100 (respectively), while displaying the message "Enter a new value..." at the top of the keypad:

```
TagnameX="MyIntegerTag"; {assign the string MyIntegerTag (which
is actually the tagname to be modified) to the memory message tag
TagnameX}

Min=-100; {assign the minimum value allowed for the tag to the
memory real/integer tag Min}

Max=100; {assign the minimum value allowed for the tag to the
memory real/integer tag Max}

MessageDisplay="Enter a new value..."; {assign the new message
string to the memory message tag MessageDisplay}

Errmsg=DialogValueEntry(TagnameX, Min, Max, MessageDisplay);
{quotes are not required because TagnameX was defined as a
message type tag. By assigning a discrete, integer or real
tagname to TagnameX, the function will modify that assigned
tagname}
```

DText() string

Dynamically changes a message tagname based on the value of a discrete tagname.

Syntax

```
MsgTag=DText(Discrete_Tag,OnMsg,OffMsg);
```

Parameters	Description
MsgTag	Message type tagname.
Discrete_Tag	Discrete type tagname.
OnMsg	Message that will be displayed when the value of *Discrete_Tag* equals 1 (True, On, Yes).
OffMsg	Will be the message displayed when *Discrete_Tag* equals 0 (False, Off, No).

Example

```
MessageTag=Dtext(DiscreteTag, DiscreteTag.OnMsg,
DiscreteTag.OffMsg);

If Dtext(D_Tag, “True”, “False”) == “True” THEN
   Message=”D_Tag contains a value of 1”;
ELSE
   Message=”D_Tag contains a value of 0”;
ENDIF;
```

Exp() math

Returns the result of *e* raised to a power.

Syntax

```
Result=Exp(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks The exponential of *Number* is calculated and returned to *Result*.

Example

```
Exp(1) will return 2.718...
```

The range for this function is -88.72 to 88.72.

FileCopy() system

Copies a *SourceFile* to a *DestFile*, similar to the DOS Copy command or the Copy function in Windows File.

Syntax

```
FileCopy(SourceFile,DestFile,DoneTag);
```

Parameters	Description
SourceFile	Source filename (including the full path).
DestFile	Destination filename (including the full path) or directory name (see the following examples).
DoneTag	Name of a tagname that **FileCopy()** function will use to report the *progress* of the copy procedure. This parameter must be a string indicating the *name* of the tagname (not the actual tagname itself). So if your done tagname is called Monitor, you would provide "Monitor" or *Monitor.name*, not just Monitor.

Remarks When the **FileCopy()** function is used it immediately returns a 1 if the procedure was successfully initiated. It returns a 0 if another procedure is already in progress (the new procedure could not be initiated) or -1 if there is an error. Using this return value, the initiation of the **FileCopy()** function can be monitored:

```
Status=FileCopy("C:\*.TXT", "C:\BACKUP", "Monitor");
```

Status is an integer tagname to which the 1, -1 or 0 will be written. **FileCopy()** function is performed in the background so that it will not interfere with the operation of InTouch. *DoneTag* allows the *progress* of the copy operation to be monitored by an application or a user. In this way, the user can be alerted to any errors which might occur *AFTER* the procedure was initiated. This is different than the *Status* returned above, which indicates whether the copy procedure has been successfully *initiated*.

Once the copy procedure has been successfully initiated, the value of *DoneTag* is then assigned. The value is set to 0 while the procedure is still in progress. It is set to 1 when the procedure is successfully completed or -1 if there is an error before the procedure could be completed.

Normally, the *SourceFile* and *DestFile* will be names of files. However, if a *single file* is being copied with **FileCopy()** function, the destination can be a directory:

```
FileCopy("C:\DATA.TXT", "C:\BACKUP", "Monitor");
```

This will copy the file "DATA.TXT" to a directory called "BACKUP" on the "C:\" drive. The tagname Monitor will be set to 1 when this is complete.

If the *SourceFile* contains any wildcards, however, the *DestFile* MUST be a directory (not a filename) or the function will return an error code:

```
FileCopy("C:\*.TXT", "C:\BACKUP", "Monitor");
```

This will copy all the .TXT files from the C:\ root directory to the directory C:\BACKUP. The tagname Monitor will be set to 1 when this is complete.

FileDelete() system

Deletes unnecessary or unwanted files.

Syntax `FileDelete(Filename);`

Parameter	Description
Filename	File name to be deleted.

Remarks If the file can be found and successfully deleted, the function will return 1. Otherwise, the function will return 0.

Example

```
Status=FileDelete("C:\DATA.TXT");
```

Status is a 1 if it finds a file called "DATA.TXT" in the C:\ root directory, or a 0 if it does not find the file.

FileMove() system

Similar to **FileCopy()** function except that it moves the file from one location to another instead of making a copy.

Syntax

```
FileMove(SourceFile,DestFile,DoneTag);
```

Parameters	Description
SourceFile	Source filename (including the full path)
DestFile	Destination filename (including the full path)
DoneTag	The name of a tagname that **FileMove()** function will use to report the *progress* of the move procedure. This parameter must be a string indicating the *name* of the tagname (not the actual tagname itself). So, if your done tagname is called Monitor, you would provide "Monitor" or Monitor name, not just Monitor.

Remarks

When the **FileMove()** function is used, it immediately returns a 1 if the procedure was successfully initiated. It returns a 0 if another procedure is already in progress (the new procedure could not be initiated) or a -1 if there is an error. Using this return value, the initiation of the **FileMove()** function can be monitored:

```
Status=FileMove("C:\DATA.TXT", "D:\DATA.TXT", "Monitor");
```

Status is a integer tagname to which the 1, -1 or 0 will be written. **FileMove()** function is performed in the background so that it will not interfere with the operation of InTouch. The purpose of the *DoneTag* is to allow the *progress* of the move operation to be monitored by an application or a user. In this way, the user can be alerted to any errors which might occur *AFTER* the procedure was initiated. This is different than the *Status* returned above, which indicates whether the move procedure has been successfully *initiated*.

Once the move procedure has been successfully initiated, the value of *DoneTag* is then assigned. The value is set to 0 while the procedure is still in progress. It is set to 1 when the procedure is successfully completed or -1 if there is an error before the procedure could be completed.

If the *SourceFile* and *DestFile* are located on the same drive, the function can simply change the file's directory reference (where the computer keeps the name and location of the file on disk) without actually moving any data. In this case, the move procedure will be very fast, regardless of the size of the file. If the *SourceFile* and *DestFile* are located on different drives, the time the move takes will vary with the size of the file. This is because the data must be transferred from one physical disk to another.

Example

```
FileMove ("C:\DATA.TXT","C:\BACKUP\DATA.TXT","Monitor");
```

This will move the file called "DATA.TXT" from the root directory of the "C" drive to a directory called "BACKUP." The tagname Monitor will be set to 1 when this is complete.

Note This function can also be used to rename files when the SourceFile and DestFile specify the same directory but different filenames.

```
FileMove ("C:\DATA.TXT","C:\DATA.BAK","Monitor");
```

This will rename the file "DATA.TXT" to "DATA.BAK" in C:\ root directory. The tagname Monitor will be set to 1 when this is complete.

FileReadFields() system

Reads a Comma Separated Variable (CSV) record from a specified file.

Syntax

```
FileReadFields(Filename,FileOffset,StartTag,NumberOfFields);
```

Parameters	Description
Filename	Specifies the file to read from.
FileOffset	Specifies the location in the file to start reading.
StartTag	Specifies the name of an InTouch tagname where the *first* data item will be written to. The name of this tagname must end with a number (such as MyTag1). This parameter must be a string indicating the *name* of the tagname (not the actual tagname itself). So, if your tagname is called MyTag1, you would provide "MyTag1" or MyTag1.name, not just MyTag1.
NumberOfFields	Specifies the number of fields to read (the number of comma-separated fields in each record in the file).

Remarks

If *StartTag* is "MyTag1" and *NumberOfFields* is 3, then 3 fields are read from the file and are stored in MyTag1, MyTag2 and MyTag3. These tags with consecutive names must be first created in InTouch and may be of different types (Integer, Message, etc.).

Examples

If the first line of C:\DATA\FILE.CSV is:

```
This is text, 3.1416, 5
```

The following script will read this line and store "This is text" into MyTag1, 3.1416 into MyTag2 and 5 into MyTag3:

```
BytePosition=FileReadFields ("C:\DATA\FILE.CSV", 0, "MyTag1", 3);
```

The function returns the new byte position after the read. You can use this return value as the *FileOffset* for the next read.

```
FileReadFields ("C:\DATA\FILE.CSV",BytePosition,"MyTag1",3);
```

Note Message Tags in InTouch can store a maximum of 131 characters.

FileReadMessage() system

Reads a specified number of bytes (or a whole line) from a specified file.

Syntax **FileReadMessage**(*Filename,FileOffset,Message_Tag,CharsToRead*);

Parameters	Description
Filename	Specifies the file to read from.
FileOffset	Specifies the location in the file to start reading.
Message_Tag	Specifies where to store the data read from the file. A maximum of 131 characters can be stored.
CharsToRead	Specifies how many bytes to read from the file. For processing text files, *CharsToRead* can be set to 0. The function will then read up to the next LF (linefeed) in the file, or up to 131 characters which ever comes first.

Example

```
FileReadMessage ("C:\DATA\FILE.TXT", 0, MsgTag, 0);
```

The first line will be read from file "C:\DATA\FILE.TXT" and stored into MsgTag. The function returns the new byte position after the read. You can use this return value as the *FileOffset* for the next read.

FileWriteFields() system

Writes a Comma Separated Variable (CSV) record to a specified file.

Syntax

```
FileWriteFields(Filename,FileOffset,StartTag,NumberOfFields);
```

Parameters	Description
Filename	Specifies the file in which to write. If *Filename* does not exist, it will be created.
FileOffset	Specifies the location in the file to start writing. If FileOffset is -1, the function will write to the end of the file.
StartTag	Specifies the name of an InTouch tagname where the first data item will come from. The name of this tagname must end with a number (such as MyTag1). This parameter must be a string indicating the name of the tagname (not the actual tagname itself). So, if your tagname is called MyTag1, you would provide "MyTag1" or MyTag1.name, not just MyTag1.
NumberOfFields	Specifies the number of fields to write (the number of comma-separated fields in each record in the file).

Remarks

If *StartTag* is "MyTag1" and *NumberOfFields* is 3, then 3 fields are written to the file (from MyTag1, MyTag2 and MyTag3). These tags with consecutive names must first be created in InTouch and may be of different types (Integer, Message, etc.).

The following script will write the line "This is text, 3.1416, 5" to the first line of C:\DATA\FILE.CSV. "This is text" is the value currently in MyTag1, 3.1416 is in MyTag2 and 5 is in MyTag3 :

```
FileWriteFields ("C:\DATA\FILE.CSV", 0, "MyTag1", 3);
```

The function returns the new byte position after the write. You can use this return value as the *FileOffset* for the next write.

The following script will write the text string MyTag1 to the end of C:\DATA\FILE.CSV:

```
FileWriteFields ("C:\DATA\FILE.CSV", -1, "MyTag1", 3);
```

FileWriteMessage() system

Writes a specified number of bytes (or a whole line) to a specified file.

Syntax

```
FileWriteMessage(Filename,FileOffset,Message_Tag,LineFeed);
```

Parameters	Description
Filename	Specifies the file to write to. If *Filename* does not exist, it will be created.
FileOffset	Specifies the location in the file to start writing. If *FileOffset* is -1, the function will write to the end of the file.
Message_Tag	Specifies the characters to write to the file.
LineFeed	Specifies whether or not to add a linefeed after the write operation. When writing to a text file, set the *LineFeed* to 1.

Remarks

The function returns the new byte position after the write. You can use this return value as the **FileOffset()** function for the next write.

Example

The following statement writes a message tagname called *MsgTag* to the end of file *C:\DATA\FILE.TXT*:

```
FileWriteMessage ("C:\DATA\FILE.TXT", -1, MsgTag, 1);
```

GetNodeName() system

Returns the NetDDE node name to a string variable.

Syntax

```
GetNodeName(Tagname,NodeNum);
```

Parameters	Description
Tagname	InTouch message type tagname that will hold the node name.
NodeNum	Integer that specifies the character length for the message tagname.

Remarks

When this script executes, the **GetNodeName()** function will read the local node name and return it to *Tagname*. (*NodeNum* sets the character length for the message tagname.)

Note This function only works if **NetDDE** is running in Windows 95 or if Network DDE is running in Windows NT.

Example

```
GetNodeName("MyNodeTag",131);
If MyNodeTag == “Master” THEN
   MessageTag = “This is the Primary Machine!”;
ENDIF;
```

GetPropertyD() GOT

Retrieves the specified property's discrete value during runtime.

Syntax

```
[ErrorNumber=]GetPropertyD("ControlName.Property",Tagname);
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

Tagname	A valid InTouch tagname (of the same type to be returned) that will hold the property value when the function is processed.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

GetPropertyI() GOT

Retrieves the specified property's integer value during runtime.

Syntax

```
[ErrorNumber=]GetPropertyI("ControlName.Property",Tagname);
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

Tagname	A valid InTouch tagname (of the same type to be returned) that will hold the property value when the function is processed.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

GetPropertyM() GOT

Retrieves the specified property's message value during runtime.

Syntax

```
[ErrorNumber=]GetPropertyM("ControlName.Property",Tagname);
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

Tagname	A valid InTouch tagname (of the same type to be returned) that will hold the property value when the function is processed.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Hide misc

Hides various windows from within a script. A **Hide()** function must precede the name of each window to be hidden.

Syntax

```
Hide Window;
```

Parameter	Description
Window	A window name or message type tagname that contains a window name.

Remarks

During runtime, if the window does not exist, the statement is ignored. If the script is only hiding or showing windows, it is best to use the Touch Pushbutton links Show Window or Hide Window instead of this function.

Example

```
Hide "My Popup Alarm Window";
```

HideSelf misc

Hides the currently active window.

Syntax

```
HideSelf;
```

Remarks

This function can only be used in an action pushbutton script.

HTGetLastError() historical

Determines if there was an error during the last retrieval of a specified pen.

Syntax

```
[Result=]HTGetLastError(Hist_Tag,UpdateCount,PenNum);
```

Parameters	Description
Hist_Tag	Historical Trend tagname representing the name of the trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
PenNum	Integer tagname or value representing the pen number (from 1-8).
[Result=]	The following results may be returned:

Result	Description
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 Name Tag Does Not Exist in Database
7	Pen # Passed to Function is invalid (not in range of 1 to 8).

Example

The following statement retrieves the error for the last retrieval of *Pen3* of the trend with the tagname *Trend1* and places the result in the integer tagname *ResultCode*:

```
[ResultCode=]HTGetLastError("Trend1",Trend1.UpdateCount,3);
```

In an animation Analog Value Display script the following statement would be used:

```
HTGetLastError("Trend1",Trend1.UpdateCount,3);
```

HTGetPenName() historical

Returns the tagname of the tagname currently used for the specified pen # of the specified trend.

Syntax

```
MessageResult=HTGetPenName(Hist_Tag,UpdateCount,PenNum);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
PenNum	Integer tagname or value representing the pen number (from 1-8). A message tagname is returned representing the specified pen's tagname.

Example

The following statement retrieves the tagname for *Pen2* of the trend with the tagname *75* and places the result in the message tagname *TrendPen*:

```
TrendPen=HTGetPenName("Trend1",Trend1.UpdateCount,2);
```

HTGetTimeAtScooter() historical

Returns the time in seconds since 00:00:00 hours GMT, January 1, 1970 for the sample at the scooter location specified by *ScootNum* and *ScootLoc*.

Syntax

```
IntegerResult=HTGetTimeAtScooter(Hist_Tag,UpdateCount,
ScootNum,ScootLoc);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of a trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
ScootNum	Integer representing the left or right scooter (1= Left Scooter, 2= Right Scooter).
ScootLoc	Real number representing the trend's **.ScooterPosRight** or **.ScooterPosLeft** field.

Remarks

UpdateCount, *ScootNum*, and *ScootLoc* cause the expression to be evaluated when any of these parameters change. This ensures that the expression is evaluated after new retrievals or after a scooter moves.

Example

The following statement retrieves the time in seconds for the value at the current scooter location for the left scooter of the trend labeled *Trend1*:

```
HTGetTimeAtScooter("Trend1",Trend1.UpdateCount,1,
Trend1.ScooterPosLeft);
```

HTGetTimeStringAtScooter() historical

Returns the string containing the time/date for the sample at the scooter location specified by *ScootNum* and *ScootLoc*.

Syntax

```
MessageResult=HTGetTimeStringAtScooter(Hist_Tag,
UpdateCount,ScootNum,ScootLoc,Format_Text);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field
ScootNum	Integer representing the left or right scooter (1=Left Scooter, 2=Right Scooter).
ScootLoc	Real number representing the trend's **.ScooterPosRight** or **.ScooterPosLeft** field.
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

UpdateCount, *ScootNum*, and *ScootLoc* cause the expression to be evaluated when any of these parameters change. This ensures that values are updated after new retrievals or after a scooter moves. The format of the string determines the contents of the return value.

Example

The following statement retrieves the date/time for the value at the current scooter location for the right scooter of the trend labeled *Trend1*. The value is stored in the message tagname *NewRightTimeString* and is in *"Time"* format:

```
NewRightTimeString=HTGetTimeStringAtScooter
("Trend1",Trend1.UpdateCount,2,Trend1.ScooterPosRight,"Time");
```

HTGetValue() historical

Returns a value of the requested type for the entire trend's specified pen. The UpdateCount parameter will cause the expression to be evaluated after a retrieval is complete.

Syntax

```
RealResult=HTGetValue(Hist_Tag,UpdateCount,PenNum,ValType_Text);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
PenNum	Integer tagname or value representing the pen number (from 1-8).
ValType_Text	String indicating the type of value to return:

Type	Description
"PenAverageValue"	Average for entire trend.
"PenMaxValue"	Max for entire trend.
"PenMinValue"	Min for entire trend.
"PenMaxEU"	Max Engineering Units for entire trend.
"PenMinEU"	Min Engineering Units for entire trend.
"PenStdDev"	Standard Deviation for entire trend.

Note When the *ValType_Text* parameter is used with the **HTGetValue** function, make sure that one of the valid types listed above is used.

Remarks

A memory real tagname is returned representing the calculated value of the given type.

Example

The following statement obtains the standard deviation for the data retrieved for the trend, *Trend1*, *Pen*2. The value is stored in the memory real tagname *LeftHemisphereSD*:

```
LeftHemisphereSD=HTGetValue("Trend1",Trend1.UpdateCount,
2,"PenStdDev");
```

HTGetValueAtScooter() historical

Returns a value of the requested type for the sample at the specified scooter position, trend and pen #. The *UpdateCount* parameter will cause the expression to be evaluated after a retrieval is complete.

Syntax

```
RealResult=HTGetValueAtScooter(Hist_Tag,UpdateCount,ScootNum,
ScootLoc,PenNum,ValType_Text);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
ScootNum	Integer representing the left or right scooter (1=Left Scooter, 2=Right Scooter).
ScootLoc	Real number representing the trend's **.ScooterPosRight** or **.ScooterPosLeft** field.
PenNum	Integer tagname or value representing the pen number (from 1-8).
ValType_Text	String indicating the type of value to return::

Type	Description
"PenValue"	Value at scooter position.
"PenValid"	0 if value is invalid, 1 if valid.

Note When the *ValType_Text* parameter is used with the **HTGetValueAtScooter** function, make sure that one of the valid types listed above is used.

Remarks

A memory real tagname is returned representing the calculated value for "PenValue." A memory discrete tagname is returned representing the value for "PenValid."

Example

The following statement stores a 1 if the value is an actual sample or a 0 if the value is invalid for *Pen*3 of trend *Trend1* in the memory discrete tagname *ValidFlag* for the right scooter's current position:

```
ValidFlag=HTGetValueAtScooter("Trend1",Trend1.UpdateCount,2,
Trend1.ScooterPosRight,3,"PenValid");
```

HTGetValueAtZone() historical

Returns a value of the requested type for the data contained between the right and left scooter positions for a trend's specified pen. The *UpdateCount* parameter will cause the expression to be evaluated after a retrieval is complete.

Syntax

```
RealResult=HTGetValueAtZone(Hist_Tag,UpdateCount,Scoot1Loc,
Scoot2Loc,PenNum,ValType_Text);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of a trend.
UpdateCount	Integer representing the trend's **.UpdateCount** field.
Scoot1Loc	Real representing the trend's **.ScooterPosLeft** field.
Scoot2Loc	Real representing the trend's **.ScooterPosRight** field.
PenNum	Integer tagname or value representing the pen number (from 1-8).
ValType_Text	String indicating the type of value to return:

Type	Description
"PenAverageValue"	Average for zone between the right and left scooter.
"PenMaxValue"	Max for zone between the right and left scooter.
"PenMinValue"	Min for zone between the right and left scooter.
"PenMaxEU"	Max Engineering Units for zone between the right and left scooter.
"PenMinEU"	Min Engineering Units for zone between the right and left scooter.
"PenStdDev"	Standard Deviation for zone between the right and left scooter.

Note When the *ValType_Text* parameter is used with the **HTGetValueAtZone** function, make sure that one of the valid types listed above is used.

Remarks

A memory real tagname is returned representing the calculated value of the given type. Specifying constant values for *Scoot1Loc* and *Scoot2Loc* has no effect and are only used to trigger the processing of the function for the purpose of updating display lines. The function uses the trend tag's **.ScooterPosLeft** and .**ScooterPosRight** directly from the runtime database for determining zone boundaries.

Example

The following statement calculates the average value for the data between the right and left scooters of trend *"Trend1"*, *Pen1*. The value is stored in the memory real tagname *AvgValue*:

```
AvgValue=HTGetValueAtZone("Trend1",Trend1.UpdateCount,
Trend1.ScooterPosLeft,Trend1.ScooterPosRight,1,
"PenAverageValue");
```

HTScrollLeft() historical

Sets the start time of the trend to a value older 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 left by a given percentage.

Syntax

```
HTScrollLeft(Hist_Tag,Percent);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
Percent	Real number representing the percent of the chart to scroll (0.0 to 100.0)

Example

The following statement scrolls the time/date for a trend with a tagname of *Trend1* by 10%:

```
HTScrollLeft("Trend1",10.0);
```

If the current display starts at 12:00:00 PM and the display width is 60 seconds, then, the new trend will start at 11:59:54 AM (after the function is processed)

HTScrollRight() historical

Sets the start time of the trend to a value newer 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 given percentage.

Syntax

```
HTScrollRight(Hist_Tag,Percent);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
Percent	Real number representing the percentage of the chart to scroll (0.0 to 100.0)

Example

The following statement scrolls the time/date for a trend with a tagname of *Trend1* by 20%:

```
HTScrollRight("Trend1",20.0);
```

If the current display starts at 12:00:00 PM and the display width is 60 seconds, then, the new trend will start at 12:00:12 PM (after the function is processed).

HTSelectTag() misc

Displays the **Select Tag** dialog box and the operator can select a different tagname for specified pen. (The dialog box only lists the tagnames that have been defined for historical logging (**Log Data** option selected) in the Tagname Database.

Syntax

```
HTSelectTag();
```

Remarks

HTSelectTag() shows ALL tags which have the Log Data option checked. However, it is possible to use the Browser's filter to display a smaller set of tags. For example all tags which begin with A. It is NOT possible however, to use the **HTSelectTag()** function to display the entire tagname database, only historical tags will be displayed.

Examples

The following script can be used in a touch pushbutton action script. The script will cause the tag browser to appear in WindowViewer. The user can then select a tagname from the list. This tagname will be used as pen 1 by the Historical Object named HistTrend.

```
HTSetPenName( "HistTrend", 1, HTSelectTag( )  );
```

See Also

HTSetPenName()

HTSetPenName() historical

Assigns a different tagname to a trend's pen.

Syntax

```
HTSetPenName(Hist_Tag,PenNum,Tagname);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
PenNum	Integer tagname or value representing the pen number (from 1-8) and *Tagname* is a string representing the new tagname to assign to the pen.
Tagname	String representing the new tagname to assign to the pen.

Remarks

This script function is the only method to add tags from a distributed history provider during runtime.

Examples

In the following statement *Pen3* of *Trend1* will use *OutletPressure* as the new tagname.

```
HTSetPenName("Trend1",3,"OutletPressure");
```

In the following statement *TrendPen4* of *Trend1* will use the distributed tagname *HistPrv1.Tag1* as the tagname.

```
HTSetPenName("Trend1",TrendPen4,"HistPrv1.Tag1");
```

See Also

HTSelectTag();

HTUpdateToCurrentTime() historical

Causes the data to be retrieved and displayed with an end time equal to the current time. The start time will be equal to EndTime minus the Width of the chart.

Syntax

```
HTUpdateToCurrentTime(Hist_Tag);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.

Example

The following statement retrieves and displays data for the historical tagname *"Trend1"* at the current time:

```
HTUpdateToCurrentTime("Trend1");
```

If it is now 3:04 PM and the width of the trend is 60 seconds, the new end time will be 3:04 PM. The new start time will be 3:03 PM.

HTZoomIn() historical

Calculates a new chart width and start time. If the trend's **.ScooterPosLeft** is 0.0 and the **.ScooterPosRight** is 1.0, then the new chart width equals the old chart width divided by two. The new start time will be calculated based on the value of *LockString*.

Syntax

```
HTZoomIn(Hist_Tag,LockString);
```

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
LockString	String representing the type of zoom:

Type	Description
"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

If the scooter positions are not at the ends, the new chart width is the time between **.ScooterPosLeft** and **.ScooterPosRight**. In this case, the value of *LockString* is not used. The minimum chart width is 1 second. The scooter positions will be 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 maintain the same start time for the trend tagname *"Trend1."* 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 (after zooming), the start time will still be 1:25:00. But, the chart width will be 15 seconds.

```
HTZoomIn("Trend1","StartTime");
```

HTZoomOut() historical

Calculates a new chart width and start time. The new chart width is the old chart width multiplied by two. The new start time will be calculated based on the value of *LockString*.

Syntax **HTZoomOut**(*Hist_Tag,LockString*);

Parameters	Description
Hist_Tag	HistTrend tagname representing the name of the trend.
LockString	String representing the type of zoom:

Type	Description
"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 scooter positions have no effect on **HTZoomOut** but will be 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 maintain the same center time for the trend tagname *"Volume."* If the start time before zooming was 2:15:00 PM and the chart width was 30 seconds, the start time after zooming will still be 2:14:45. The chart width will be 60 seconds and the center of the trend will remain at 2:15:15.

```
HTZoomOut("Volume","Center");
```

InfoAppActive() system

Tests whether an application is active.

Syntax DiscreteResult=**InfoAppActive**(*AppTitle*);

Parameters	Description
AppTitle	String representing application to report on.

Example

```
InfoAppActive("Microsoft Excel") will return 1 {if it is running}
InfoAppActive("Calculator") will return 0 {if it is not running}
```

Note The application title for a particular application can be determined using the **InfoAppTitle()** function.

The following script example uses **InfoAppActive** to inspect the list of tasks running on the system. If a task called Notepad is running it will return a result of 1. Thus one can avoid launching a second instance of Notepad. If **InfoAppActive** returns a value of 0 (Notepad is not already running) then Notepad can be launched.

```
IF InfoAppActive( InfoAppTitle( "Notepad" ) ) == 1 THEN
   ActivateApp  InfoAppTitle( "Notepad" );
ELSE
   StartApp  "Notepad";
ENDIF;
```

InfoAppTitle() system

Returns the Application Title or Windows Task list name of a specified program which is currently running.

Syntax

```
MessageResult=InfoAppTitle(“ProgramEXEName”);
```

Parameters	Description
ProgramEXEName	String representing program to report on.

Example

The following statement will process *ProgramEXEName*, "calc", to return "Calculator." The actual program name is calc.exe. Do not use the ".exe" in the Program EXE name string.

```
InfoAppTitle("calc") will return "Calculator"
InfoAppTitle("excel") will return "Microsoft Excel"
```

InfoDisk() system

Returns information about a specific local (or network) disk drive.

Syntax

```
IntegerResult=InfoDisk("Drive",InfoType,Trigger);
```

Parameters	Description
Drive	String or message tagname representing drive letter.
InfoType	Integer representing information type:

Type	Description
1	Returns the total size (in bytes) of the disk drive.
2	Returns the available space free (in bytes) on the disk drive.
3	Returns the total size (in kilobytes) of the disk drive.
4	Returns the available space free (in kilobytes) on the disk drive.

Trigger	Executes the **InfoDisk** function every time the value of *Trigger* changes. *Trigger* can be any tagname (not limited to system variables). This parameter is used for expression fields in animation links only, any value can be used as a place holder when **InfoDisk()** is used in a script, because this parameter has no effect in a script.

Remarks Information about the disk drive specified by *Drive* is returned to *IntegerTag*.

Example The following statement executes every minute and returns the current value. If used in a value display analog animation link it will refresh once per minute.

```
InfoDisk("C", 1, $Minute) will return 233869345 {total size in bytes}
InfoDisk("C", 2, $Minute) will return 3238935 {free space in bytes}
InfoDisk("C", 3, $Minute) will return 228388 {total size in kilobytes}
InfoDisk("C", 4, $Minute) will return 3163 {free space in kilobytes}
```

Comment

```
1 kilobyte = 1024 bytes
```

Note As in other functions where single characters are used, if the message tagname (used for *Drive*) provided to **InfoDisk()** function contains more than one character, only the first character of the tagname will be used. Since the function relies upon information returned from the operating system, the values may be erroneous on Windows 95 operating systems containing drives which are larger than 2 gigabytes.

InfoFile() system

Returns information about a specific file or subdirectory.

Syntax

```
IntegerResult=InfoFile("Filename","InfoType",Trigger);
```

Parameters	Description
Filename	String representing filename to act on actual string or message tagname.
InfoType	Integer representing type of information to retrieve:

Type	Description
1	*Does the file exist?* Returns 1 if the filename is an actual file. Returns 2 if the filename is a sub-directory. Returns 0 if the function cannot find the file.
2	File size (in bytes).
3	File Date/Time (in seconds since Jan-1-1970).
4	Number of files that match the filename description. Values greater than 1 will be returned only in wildcard searches in which more than one file matching the search criterion is found.

Trigger	Any tagname. *Trigger* will execute the **InfoFile** function every time the value of *Trigger* changes. This parameter is used for expression fields in animation links only, any value can be used as a place holder when **InfoFile()** is used in a script, because this parameter has no effect in a script.

Remarks

Information about the file specified by *Filename* is returned to *IntegerTag*. The *Filename* has to include the full path name to the file, but can include wildcard characters (*, ?).

Example

The following statement executes every minute and return the following values:

```
InfoFile("c:\IT56\view.exe", 1, $Minute) will return 1
{file found}

InfoFile("c:\InTouch\view.exe", 2, $Minute) will return 634960
{file size}

InfoFile("c:\InTouch\view.exe", 3, $Minute) will return 736701852
{secs since 1-1-70}

InfoFile("c:\InTouch\*.exe", 4, $Minute) will return 17
{found 17 EXE files}
```

InfoInTouchAppDir() system

Returns the current InTouch application directory.

Syntax

```
MessageResult=InfoInTouchAppDir();
```

Remarks

The current InTouch application directory is returned into *MessageResult*.

Example

```
InfoInTouchAppDir() will return "c:\InTouch.32\demoapp1"
```

InfoResources() system

Returns various system resource values.

Syntax

```
IntegerResult=InfoResources(ResourceType,Trigger);
```

Parameters	Description
ResourceType	Integer representing resources to monitor:
Trigger	Executes the **InfoResources()** function every time the value of *Trigger* changes. *Trigger* can be any tagname (not limited to system variables). This parameter is used for expression fields in animation links only, any value can be used as a place holder when **InfoResources()** is used in a script, because this parameter has no effect in a script.

Type	Description
1	Returns the percentage of free space for GDI resources.
2	Returns the percentage of free space for USER resources.
3	Returns the number of bytes of memory currently free
4	Returns the number of tasks currently running.

Remarks

The particular system resource specified by the integer *ResourceType* is stored in *IntegerResult*.

Example

The following statement executes every minute and return the current value:

```
InfoResources(1, $Minute) will return 54  {% free}
InfoResources(2, $Minute) will return 36  {% free}
InfoResources(3, $Minute) will return 11524093  {bytes}
InfoResources(4, $Minute) will return 14  {tasks}
```

Example

System resource values:

Case 1 and Case 2:
GDI and USER are hard-coded to return '50%' on Windows NT and Windows 95.

Case 3:
On Windows NT and Windows 95 returns "free bytes of paging file."

Case 4:
On Windows NT and Windows 95 returns the result of search of all the top level windows. It only counts the windows that are visible and do not have any owners. This is not the actual "number of tasks currently running" in the system. Its closest approximation would be the count of items on **Applications** tab when you run the **Task Manager** in Windows NT or the **Close Program** window which comes up when you press CTRL+ALT+DEL in Windows 95.

Int() math

Returns the next integer less than or equal to a specified number.

Syntax

```
IntegerResult=Int(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks

When dealing with negative real numbers, this function will return the integer farthest from zero.

Example

```
Int(4.7) will return 4
Int(-4.7) will return -5
```

IOSetAccessName() misc

Modifies the *application* or *topic* name portions of a Access Name during runtime which allows implementing of hot-backup strategies for InTouch.

Syntax

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

Parameters	Description
AccessName	The existing Access Name to assign the new AppName and Topic Name values to. Actual string or message tagname.
AppName	The new Application Name to assign. Actual string or message tagname.
TopicName	The new Topic Name to assign. Actual string or message tagname.

Remarks

The Access Name, Application Name and Topic Name values may be specified as literal strings, or they may be string values provided by other InTouch tags or functions. For example, the Access Name *MyAccess1* can be changed to point to the "EXCEL" application and the “Sheet1” topic by:

Example

```
IOSetAccessName("MyAccess1", "EXCEL", "Sheet1");
```

or by

```
Number = 1;
AccNameString = "MyAccess" + Text(Number, "#");
IOSetAccessName(AccNameString, "EXCEL", "Sheet1");
```

If an empty string is specified for a Topic, then the Access Name’s current Application value is retained and its Topic value is updated. For example, the following changes the Application Name for access name MyAccess2 to "excel" without affecting its current Topic value:

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

Likewise, if an empty string is specified only for an Application Name, then the tag’s current Topic value is retained and its Application value is updated. For example, the following changes the Topic for tagname MyAccess3 to "Sheet2" without affecting its current Application Name value:

```
IOSetAccessName("MyAccess3", "", "Sheet2");
```

Note When **IOSetAccessName()** is processed, there is a time delay as the existing conversation is terminated and the new conversation is initiated. During this time, any attempted POKEs or writes to the new topic will be lost.

IOSetItem() misc

Changes the access name and/or item in a I/O type tagname **.Reference** field.

Syntax

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

Parameters	Description
Tagname	Any InTouch IO tagname enclosed in quotes.
AccessName	The AccessName to change to the tagname to.
Item	The Item to change the tagname to.

Examples

```
IOSetItem(TagName, AccessName, Item)
```

The Tagname, AccessName, and Item values may be specified as literal strings, or they may be string values provided by other InTouch tags or functions. For example, the **.Reference** field of tagname "MyTag1" can be changed to point to the "excel" access name and the "R1C1" item by:

IOSetItem("MyTag1", "excel", "R1C1");

or by

Number = 1;
TagNameString = "MyTag" + Text(Number, "#");
IOSetItem(TagNameString, "excel", "R1C1");

If an empty string ("") is specified for both the Access Name and Item values, then the tagname is deactivated. For example, the tagname MyTag2 is deactivated by:

IOSetItem("MyTag2", "", "");

If an empty string is specified only for an Item, then the tag's current Item value is retained and its AccessName value is updated. For example, the following changes the Access Name for tagname MyTag3 to "excel2" without affecting its current Item value:

IOSetItem("MyTag3", "excel2", "");

Likewise, if an empty string is specified only for an AccessName, then the tag's current Item value is retained and its AccessName value is updated. For example, the following changes the Item for tagname MyTag3 to "R1C2" without affecting its current Access Name value:

IOSetItem("MyTag4", "", "R1C2");

IsAnyAsynchFunctionBusy() system

Used to find out if any asynchronous QuickFunctions are running. This function can be used to make the QuickScript that calls an asynchronous QuickFunction wait for all other asynchronous QuickFunctions to complete processing. This allows the QuickScript to re-synchronize itself.

Syntax

```
DiscreteTag=IsAnyAsynchFunctionBusy(timeout);
```

Parameters	Description
DiscreteTag	A discrete type tagname to which a value is returned as follows: If the function times out, waiting all executing QuickFunctions to complete, a 1 (true) is returned to *DiscreteTag*. If there are no asynchronous QuickFunctions running, a value of 0 (false) will be returned immediately, or the QuickFunction will wait for the timeout to elapse. It will also return a 0 if after timing out there are no asynchronous QuickFunctions running.
timeout	An integer value representing the number of seconds to wait to determine if any asynchronous QuickFunctions are running.

Example

Assume you want to connect to several SQL databases using asynchronous QuickFunctions and, you know that it will take 2 minutes to make those connections. First, you would execute the asynchronous QuickFunctions to connect to the SQL databases. Next, you would launch the function, **IsAnyAsynchFunctionBusy(120)** to allow enough time for SQL to make the connections before completing the QuickFunction.

However, if after 2 minutes the connections have not been made and the asynchronous QuickFunctions are still busy trying to make the connections, a value of 1 (true) will be returned by the **IsAnyAsynchFunctionBusy()** function. You can now display an error message telling the operator that the SQL connections were unsuccessful.

Use the following window **On Show** QuickScript:

```
IF IsAnyAsynchFunctionBusy(120) == 1 THEN
   SHOW "SQL Connection Error Dialog";
ENDIF;
```

Log() math

Returns the natural log of a number.

Syntax

```
RealResult=Log(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks

The natural log of *Number* is calculated and returned to *RealResult*. Natural log of 0 is undefined.

Example

```
Log(100) will return 4.605...
Log(1) will return 0
```

LogMessage misc

Writes a user-defined message to the Wonderware Logger.

Syntax

```
LogMessage("Message_Tag");
```

Parameter	Description
Message_Tag	String to log to Wonderware Logger. Actual string or message tagname.

Remarks

This is a very powerful function for troubleshooting InTouch scripting. By strategically placing **LogMessage()** functions in your scripts you can determine the order of script execution, performance of scripts and identify the value of tags both before they are changed and after they have been effected by the script. Each message posted to the Wonderware Logger is stamped with the exact date and time.

Example

```
LogMessage("Report Script is Running");
```

The above statement would print the following to the Wonderware Logger:
94/01/14 15:21:14 WWSCRIPT Message:Report Script is Running.

```
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
MyTag+MyTag + 10;
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

LogN() math

Returns the values of the *logarithm* of *x* to base *n*.

Syntax

```
Result=LogN(Number,Base);
```

Parameter	Description
Number	Any number, real or integer tagname.
Base	Integer to set log base. Number or Integer tagname.

Remarks Base 1 is undefined.

Example

```
LogN(8, 3) will return 1.89279...
```

```
LogN(NumberTag, BaseTag) will return 0.564...if NumberTag
contains 3 and BaseTag contains 7.
```

Pi() math

Returns the value of *Pi*.

Syntax

```
RealResult=Pi();
```

Example

```
Pi() will return 3.1415926...
```

PlaySound() misc

Plays a wave form sound specified by a .wav filename or an entry in the **[sounds]** section of the WIN.INI file through the Windows sound device (if installed).

Syntax

```
PlaySound(“SoundName”,Flags);
```

Parameter	Description
SoundName	String or message tagname that represents sound file to play.
Flags	*Flags* can be one of the following:

Type	Description
0	Play sound synchronously (default)
1	Play sound asynchronously
2	Do not use default sound. Playsound is the filename of a .wav file. PlaySound can also accept an entry name from the [Sounds] section of the WIN.INI file.
3	NOT USED!
4	Sound points to a memory file. This flag is not useful within InTouch scripting.
5-7	NOT USED!
8	Repeat the sound until the next time **PlaySound()** is called.

Example

```
PlaySound ("c:\horns.wav",1);
```

Note The sound must fit in available physical memory and be playable on an installed wave form audio device driver. The directories searched for sound files are in order: the current directory; the windows directory; the windows system directory; the directories listed in the PATH. If a specified .wav sound cannot be found, go to your **Start** menu, point to **Settings**, click **Control Panel**, then double-click on the **Sounds** icon, setup your default sound. If the default sound cannot be found, no sound is played.

PrintHT() historical

Can be used in a button for printing the Historical Trend chart associated with the specified Hist Trend type tagname. The Historical Trend must be visible on the screen when using this function.

Syntax

```
PrintHT("Trend_Tag");
```

Parameter	Description
Trend_Tag	History Trend tagname. Actual string or message tagname.

Remarks None

Example

```
PrintHT("HistTrend1");
```

PrintWindow() misc

Prints the specified window.

Syntax

```
PrintWindow("Window",Left,Top,Width,Height,Options);
```

Parameters	Description
Window	The name of the window to print. Actual string or message tagname.
Left	A floating point number whose units are inches to set the left margin. Number or real tagname.
Top	A floating point number whose units are inches to set the top margin. Number or real tagname.
Width	A floating point number whose units are inches that sets the width of the printout. This can be 0 to default to largest aspect ratio or can be a specific width. Number or real tagname.
Height	A floating point number whose units are inches that set the height of the printout. This can be 0 to default to largest aspect ratio or can be a specific height. Number or real tagname.
Options	A discrete value, 0 or 1, that is only used if *Width* and *Height* are 0. If *Options* is 1, the window is printed with the largest aspect ratio that is an integer multiple of the window size. Discrete number or discrete tagname. If *Options* is 0, the window is printed with the largest aspect ratio that fits on the page.

Note If the window contains a bitmap, set *Options* to 1 to prevent the bitmap from being stretched.

Remarks

Many reports can be queued using this function. (It is recommended that **PrintWindow()** function be used in place of the **PrintHT()** function and user initiated prints from the Historical Trend runtime dialog box whenever the entire screen is to be printed, not just the trend chart.)

When this function is processed, View loads the window in an "off screen" memory area. View then waits 10 seconds to allow all of the DDE variables to be updated. The window is then sent to the printer. The window specified does not need to be open or visible to be printed. The amount of time View waits can be controlled by adding the following line to the INTOUCH.INI file:

```
PrintWindowWait=10000
```

10000 represents the number of milliseconds to wait.

Fonts are printed as fonts, objects are bit mapped and printed as such. Windows with white backgrounds that contain only fonts are printed very fast. Windows with colored backgrounds that contain many objects will take longer to print.

Note To ensure that text in a window prints properly, it is recommended that "True Type" fonts be used for the text fields in all windows that will be printed.

When printing pushbuttons, the text on the button may appear "cut off" because the font used for the button text is the "System" font, which is not a "True Type" font. And, the "System" font in the printer is slightly different than the "System" font used on the screen. If this should occur, try expanding the button.

If the printer is being used to print alarms, a second printer is required to use the **PrintWindow()** function.

Examples

The following On True Condition Script prints a three page report every day at 8:30 AM:

Condition:

```
$Hour == 8 AND $Minute == 30
```

Script Body:

```
PrintWindow("1st Shift Summary",1,1,0,0,0);
PrintWindow("2nd Shift Summary",1,1,0,0,0);
PrintWindow("3rd Shift Summary",1,1,0,0,0);
```

The **PrintWindow()** function returns a 1 if the window name exists and could be queued to print. Otherwise, it returns 0. Therefore, status of the function can be monitored.

```
Status=PrintWindow("Shift Summary",1,1,0,0,0);
```

Status is a discrete tagname to which the 1 or 0 will be written.

RecipeDelete() recipe

Deletes a Recipe Name from the specified recipe template file.

Syntax

```
RecipeDelete("Filename","RecipeName");
```

Parameters	Description
FileName	*Recipe template file* that will be acted upon by the function. Actual string or message tagname.
RecipeName	Specific recipe in the designated recipe template file that will be deleted by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the *RecipeName*. The **RecipeSelectRecipe()** function returns a value to this parameter. Actual string or message tagname.

Example

The following statement deletes the recipe "Recipe1" from the recfile.csv file:

```
RecipeDelete("c:\recipe\recfile.csv", "Recipe1");
```

RecipeGetMessage() recipe

Writes an error code to an analog tagname and the corresponding error code message to a message tagname.

Syntax

```
RecipeGetMessage(Analog_Tag,Message_Tag,Number);
```

Parameters	Description
Analog_Tag	Actual Integer or Real tagname with no quotes or constants.
Message_Tag	Actual Message tagname with no quotes or string literal.
Number	This parameter sets the maximum string length returned to the *Message_Tag*. InTouch, message tagnames have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of *Message_Tag* in the InTouch Tagname Dictionary. This parameter can be a constant or an integer tagname containing a number.

Example

By using the **RecipeGetMessage()** function in an InTouch Data Change script, the corresponding error code can be written to an analog tagname and the associated error code message can be written to a message tagname:

```
Data Change Script Tagname[.field]:ErrorCode

Script body:RecipeGetMessage(ErrorCode, ErrorMessage,131);
```

This script will automatically execute whenever the value of the analog tagname ErrorCode changes. When this script executes, the **RecipeGetMessage()** function will read the current numeric value of the tagname ErrorCode and return the message associated with that value to the tagname ErrorMessage.

```
ErrorCode = RecipeLoad ("c:\App\recipe.csv","Unit1","cookies");

RecipeGetMessage(ErrorCode, ErrorMessageTag, 131);
```

RecipeLoad() recipe

Loads a specific recipe to a specific unit of tagnames.

Syntax

```
RecipeLoad("Filename","UnitName","RecipeName");
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. The *FileName* can be a string constant or a message tagname that contains the name of the recipe template file.
UnitName	Name of the specific unit in the designated recipe template file that will be used by the function. The **RecipeLoad()** function requires the user to provide the UnitName. The **RecipeSelectUnit()** function returns a value to this parameter. The UnitName can be a string constant or a message tagname that contains the name of the unit.
RecipeName	Name of the specific recipe in the designated *recipe template file* that will be used by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the RecipeName. The **RecipeSelectRecipe()** function returns a value to this parameter. The **RecipeName** can be a string constant or a message tagname that contains the name of the recipe.

Example

The following statement causes the values defined for the recipe named Recipe1 (in the recfile.csv file) to be loaded into the tagnames defined for Unit:

```
RecipeLoad("c:\recipe\recfile.csv", "Unit1", "Recipe1");
```

RecipeSave() recipe

Saves a newly created recipe or to save changes made to an existing recipe to the specified recipe template file.

Syntax

```
RecipeSave("Filename","UnitName","RecipeName");
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. The *FileName* can be a string constant or a message tagname that contains the name of the recipe template file.
UnitName	Name of the specific unit in the designated recipe template file that will be used by the function. The **RecipeLoad()** function requires the user to provide the UnitName. The **RecipeSelectUnit()** function returns a value to this parameter. The UnitName can be a string constant or a message tagname that contains the name of the unit.
RecipeName	Name of the specific recipe in the designated *recipe template file* that will be used by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the *RecipeName*. The **RecipeSelectRecipe()** function returns a value to this parameter. The *RecipeName* can be a string constant or a message tagname that contains the name of the recipe.

Example

The following example will save changes made to the recipe named "Recipe3" in the recfile.csv file. If Recipe3 does not currently exist in the recfile.csv file, it will be created. The values will then set the values of the tagnames defined for Unit2:

```
RecipeSave("c:\recipe\recfile.csv", "Unit2", "Recipe3");
```

RecipeSelectNextRecipe() recipe

Selects the next Recipe Name currently defined in the recipe template file.

Syntax

```
RecipeSelectNextRecipe("Filename", RecipeName, Number);
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. Actual message tagname.
RecipeName	Name of the specific recipe in the designated *recipe template file* that will be used by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the *RecipeName*. The **RecipeSelectRecipe()** function returns a value to this parameter. Actual Message tagname with out quotes or a string literal.
Number	If a function has to fill a parameter with characters, this field sets the maximum string length returned to the parameter. In InTouch, string (message) tagnames have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tagname. Number or integer tagname.

Example

The following statement causes the system to read the current value of the tagname *RecipeName* and return the next recipe on file. If the value of *RecipeName* is blank or cannot be found, the first recipe on file is returned. If *RecipeName* currently contains the last Recipe Name on file, it is returned unchanged. (Recipes are saved in the order in which they are created.)

```
RecipeSelectNextRecipe("c:\recipe\recfile.csv", RecipeName, 131);
```

RecipeSelectPreviousRecipe() recipe

Selects the previous Recipe Name currently defined in the recipe template file.

Syntax

```
RecipeSelectPreviousRecipe("Filename",RecipeName,Number);
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. Actual message tagname.
RecipeName	Name of the specific recipe in the designated *recipe template file* that will be used by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the *RecipeName*. The **RecipeSelectRecipe()** function returns a value to this parameter. Actual Message tagname with out quotes or a string literal.
Number	If a function has to fill a parameter with characters, this field sets the maximum string length returned to the parameter. In InTouch, message tagnames have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tagname. Number or integer tagname.

Example

The following statement causes the system to read the current value of the tagname *RecipeName* and return the previous Recipe Name on file. This returned string will be stored in *RecipeName* and will overwrite the current value. If the value of *RecipeName* is blank or cannot be found, the last recipe on file is returned. If *RecipeName* currently contains the first Recipe Name on file, it is returned unchanged. (Recipes are saved in the order in which they are created.)

```
RecipeSelectPreviousRecipe("c:\recipe\recfile.csv", RecipeName,
131);
```

RecipeSelectRecipe() recipe

Selects a specific Recipe Name currently defined in the recipe template file.

Syntax

```
RecipeSelectRecipe("Filename",RecipeName,Number);
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. Actual message tagname.
RecipeName	Name of the specific recipe in the designated *recipe template file* that will be used by the function. The **RecipeLoad(), RecipeSave()** and **RecipeDelete()** functions require the user to provide the *RecipeName*. The **RecipeSelectRecipe()** function returns a value to this parameter. Actual Message tagname with out quotes or a string literal.
Number	If a function has to fill a parameter with characters, this field sets the maximum string length returned to the parameter. In InTouch, message tagnames have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tagname. Number or integer tagname.

Example

The following statement causes the **Select a Recipe** dialog box to open:

```
RecipeSelectRecipe("c:\recipe\recfile.csv", RecipeName, 131);
```

Once a recipe is selected from the dialog box, its name is returned to the tagname *RecipeName*.

RecipeSelectUnit() recipe

Selects the unit of tagnames to which the current recipe values will be loaded.

Syntax

```
RecipeSelectUnit("Filename", UnitName,Number);
```

Parameters	Description
FileName	Name of the *recipe template file* that will be acted upon by the function. Actual message tagname.
UnitName	Name of the specific unit in the designated recipe template file that will be used by the function. The **RecipeLoad()** function requires the user to provide the *UnitName*. The **RecipeSelectUnit()** function returns a value to this parameter. Actual Message tagname with out quotes or a string literal.
Number	If a function has to fill an parameter with characters, this field sets the maximum string length returned to the parameter. In InTouch, string (message) tagnames have a maximum length of 131 characters. Use 131 for this parameter unless you have reduced the maximum string length of the InTouch tagname. Number or integer tagname.

Example

The following statement causes the **Select a Unit** dialog box to open:

```
RecipeSelectUnit("c:\recipe\recfile.csv", UnitName, 131);
```

Once a Unit is selected, its name is returned to the tagname *UnitName*.

Note Both the **RecipeSelectRecipe()** and **RecipeSelectUnit()** functions are used in conjunction with the **RecipeLoad()** function.

📖 For more information on combining functions, see the "Combining Recipe Functions" section of the *Recipe Manager User's Guide*.

RestartWindowViewer system

Allows the user control over shutting down and restarting WindowViewer.

Syntax

```
RestartWindowViewer;
```

Remarks

This function will shut down and then automatically restart WindowViewer. It is used to update an application when the automatic update Network Application Development (NAD) functions are not used. This function can be used with **$ApplicationChanged** to determine when a NAD update has occurred and then provide a custom shut-down. When using the NAD option, **Notify Operator**, the operator may have to delay the update until a later time, or this function can be placed in a push button action script. Therefore an operator can perform an automatic shutdown and restart of WindowViewer when it is convenient.

See Also

$ApplicationChanged

Round() math

Rounds a real number to a specified precision.

Syntax

```
RealResult=Round(Number,Precision);
```

Parameter	Description
Number	Any number, real or integer tagname.
Precision	Sets the precision to which the number will be rounded. Number, real or integer tagname.

Remarks

The *Precision* parameter sets the precision to which the *Number* will be rounded.

Examples

```
Round(4.3, 1) will return 4
Round(4.3, .01) will return 4.30
Round(4.5, 1) will return 5
Round(-4.5, 1) will return -5
Round(106, 5) will return 105
Round(43.7, .5) will return 43.5
```

See Also

Trunc()

SendKeys misc

Sends keys to an application. To the receiving application, the keys appear to have been entered from the keyboard. This capability may be used to enter data into the application or to give commands to the application. Most keyboard keys can be used in a **SendKeys** statement. Each key is represented by one or more characters such as A for the letter A or {ENTER} for the Enter key.

Syntax

SendKeys *KeySequence*;

Parameter	Description
KeySequence	Any key sequence or message type tagname.

Remarks

To specify more than one key, concatenate the codes for each character. For example, to specify the dollar sign ($) key followed by a (b), enter $b. The following lists the valid send keys for unique keyboard keys:

Key	Code	Key	Code
BACKSPACE	{BACKSPACE} or {BS}	HOME	{HOME}
BREAK	{BREAK}	INSERT	{INSERT}
CAPSLOCK	{CAPSLOCK}	LEFT	{LEFT}
DELETE	{DELETE} or {DEL}	NUMLOCK	{NUMLOCK}
DOWN	{DOWN}	PAGE DOWN	{PGDN}
END	{END}	PAGE UP	{PGUP}
ENTER	{ENTER} or ~ (tilde)	PRTSC	{PRTSC}
ESCAPE	{ESCAPE} or {ESC}	RIGHT	{RIGHT}
F1	{F1}*	TAB	{TAB}
		UP	{UP}

* All functions keys are entered the same.

Special keys (SHIFT, CTRL and ALT) have their own key codes:

Key	Code
SHIFT	+ (plus)
CTRL	^ (caret)
ALT	% (percent)

Examples

If two special keys are to be used together, a second set of parentheses is required. The following statement is used to hold down the CTRL key while pressing the ALT key followed by p:

```
SendKeys "^(%(p))";
```

Commands can be preceded by the **ActivateApp** command to direct the keystrokes to the proper application.

The following statement gives the computer focus to Excel and send the key combination CTRL+P (which may run a previously defined print macro assigned to run with the key combination CTRL+P):

```
ActivateApp "Microsoft Excel";
SendKeys "^(p)";
```

Or the following can be used to bring up the logon dialog box in WindowViewer:

```
SendKeys "%(SYL)";
```

In a button labeled HELP you could use the following action script:

```
SendKeys "{F1}";
```

SetDDEAppTopic() misc

This function is replaced by **IOSetAccessName** beginning with InTouch Version 7.0. See **IOSetAccessName**.

For more information refer to the **IOSetAccessName** function in this user's guide.

SetDDEItem() misc

This function is replaced by **IOSetItem** beginning with InTouch Version 7.0. See **IOSetItem**.

For information refer to **IOSetItem** function in this user's guide.

SetPropertyD() GOT

Specifies the property's discrete value that is to be written during runtime.

Syntax

```
[ErrorNumber=]SetPropertyD("ControlName.Property", DiscreteTag);
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

DiscreteTag	A discrete value, or discrete tagname that holds the value to be written when the function is processed. Typical Usage: 0=Control is disabled 1= Control is enabled

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

SetPropertyI() GOT

Specifies the property's integer value that is to be written during runtime.

Syntax

```
[ErrorNumber=]SetPropertyI("ControlName.Property", Integer);
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

Integer	A number or integer tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

SetPropertyM() GOT

Specifies the property's message value that is to be written during runtime.

Syntax

```
[ErrorNumber=]SetPropertyM("ControlName.Property", "MessageTag");
```

Parameter	Description
ControlName	Name of a windows control. for example, *ChkBox_1* or the name of an alarm object. for example, *AlmObj_1*.
.Property	Windows control or alarm object property.

For more information on these properties, see Chapter 2 - Dot Fields.

MessageTag	The string that will be written to the *ControlName*'s property. Actual string or message tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Sgn() math

Determines the sign of a value (whether it is positive, zero, or negative).

Syntax

```
IntegerResult=Sgn(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks

If the input number is positive, the result will be 1. Negative numbers will return a -1 and 0 will return a 0.

Examples

```
Sgn(425) will return 1
Sgn(0) will return 0
Sgn(-37.3) will return -1
```

Show misc

Displays a specified window. (Window name must be enclosed in quotation marks.)

Syntax

```
Show "Window";
```

Parameters	Description
Window	The name of the window to display. Actual string or message tagname.

Remarks

Window must match the name of an existing window or a window that will be created. During runtime, if the window does not exist, WindowViewer ignores the statement. If the name of the window changes, it must be changed in the script as well.

Example

```
Show "Alarm Summary Window";
```

Note If the script is only hiding or showing windows, it is recommended that the **Show** *Window* or **Hide** *Window* links be used instead. If these are used and the window name changes, InTouch will automatically make the change.

ShowAt() misc

Specifies the horizontal and vertical pixel location of a window when it is shown.

Syntax

```
ShowAt("Window",Horiz,Vert);
```

Parameters	Description
Window	The name of the window. Actual string or message tagname.
Horiz	Horizontal pixel location. Number or integer tagname.
Vert	Vertical pixel location. Number or integer tagname.

Remarks

When the window opens, it will be centered on the horizontal and vertical coordinates. The window will not be centered if one of its edges extends off-screen. A best fit to that edge will be performed.

Examples

In the following statement, *100* represents the horizontal pixel location and *200* represents the vertical:

```
ShowAt("Window Name",100,200);
```

Analog Input objects can also be created and linked to memory tagnames, for example, *TagHoriz* and *TagVert*, to dynamically change the window's position during runtime. In this case, the statement would be entered as:

```
ShowAt("Boiler Room 7 Details",TagHoriz,TagVert);
```

The following statement determines the pixel locations of an object, the internal tagnames **$ObjHor** and **$ObjVer** can be assigned to analog output links that will display the locations for the currently selected object. A window can be made to appear centered on top of an object or pushbutton by using the **$ObjHor** and **$ObjVer** tagnames in a script linked to the object or pushbutton.

```
ShowAt("Window Name",$ObjHor,$ObjVer);
```

See Also

$ObjHor, $ObjVer, ShowTopLeftAt()

ShowHome misc

Displays the "home" window(s). Home windows are those you selected to open automatically when WindowViewer is started. (The home windows are selected in the **WindowViewer Properties - Home Windows** property sheet.)

Syntax

```
ShowHome;
```

ShowTopLeftAt() misc

Specifies the horizontal and vertical pixel location of the top left corner of a window when it is shown.

Syntax

```
ShowTopLeftAt("Window",Horiz,Vert);
```

Parameters	Description
Window	The name of the window. Actual string or message tagname.
Horiz	Horizontal pixel location. Number or integer tagname.
Vert	Vertical pixel location. Number or integer tagname.

Remarks

When the window opens, its top left corner will be positioned where the horizontal and vertical coordinates meet. (The uppermost left corner of the screen is pixel location 0,0.) This function works exactly the same as the **ShowAt()** function, except it controls where the top left corner of the window will appear.

See Also

ShowAT()

Sin() math

Returns the *sine* of an angle given in degrees.

Syntax

```
Result=Sin(AngleNumber);
```

Parameter	Description
AngleNumber	Angle in degrees. Any number, real or integer tagname.

Remarks

The sine of *Number* is calculated and returned to *Result*.

Example

```
Sin(90) will return 1
Sin(0) will return 0
wave = 100 * sin (6 * $second);
```

See Also

Cos(), Tan()

SPCConnect() SPC

This function is used in conjunction with automatic data collection datasets. Before an Automatic data collection dataset will start collecting data, this function must be used to indicate to SPC which user the node is.

Syntax

```
SPCConnect("User","Password");
```

Parameter	Description
User	Database user name. Actual string or message tagname.
Password	User's password. Actual sting or message tagname.

Remarks

Executing this function will connect the user to the database and start all automatic data collection datasets using this user ID. If there is no password defined for the database, you can use the following script.

Example

```
SPCConnect("User1", "");
```

SPCDisconnect() SPC

This function is used to disconnect an Agent from a SPC Pro database. When this function is used, all datasets that were assigned to the Agent that was disconnected will stop collecting.

Syntax

```
SPCDisconnect();
```

Remarks

Executing this function will disconnect the user from the database and stop all automatic data collection datasets.

Example

```
SPCDisconnect();
```

SPCDisplayData() SPC

This function is designed to allow convenient scrolling of the chart to any date and time. You can use a tagname to monitor the status of the SPC data search. Status will contain a **0** if SPC found data and **1** if it could not find data for the specified time period.

Syntax

```
[Status=]SPCDisplayData("Dataset", "DateString", "TimeString", RangeInHours );
```

Parameter	Description
Dataset	Represents an actual Dataset name. Actual string or message tagname.
DateString	Represents the date in the format mm/dd/yy. Actual string or message tagname.
TimeString	Represents the Time in the format hh:mm:ss. Actual string or message tagname.
RangeInHours	Represents the Hours of data to display. Number or integer tagname.

Example

```
StatusTag = SPCDisplayData("Dataset", "DateString",
"TimeString" , RangeInHours);
```

SPCLocateScooter() SPC

This function is designed to allow convenient scrolling of the Scooter to any valid sample number. The Scooter tagname defined in the dataset will be updated with the X-Bar sample value. Setting SampleNumber to **0** hides/disables the Scooter.

Syntax

```
SPCLocateScooter( "Dataset", SampleNumber );
```

Parameter	Description
Dataset	Represents an actual Dataset name. Actual string or message tagname.
SampleNumber	Represents any valid sample number. Number or integer tagname.

SPCMoveScooter() SPC

This function is designed to allow convenient scrolling of the Scooter to any valid sample number. The Scooter tagname defined in the dataset will be updated with the X-Bar sample value.

Syntax

```
SPCMoveScooter( "Dataset", IncrementValue );
```

Parameter	Description
Dataset	Represents an actual Dataset name. Actual string or message tagname.
IncrementValue	Represents any valid number. Positive to scroll forward and Negative to scroll backwards. Number or integer tagname.

SPCSaveSample() SPC

Saves a manually input sample. This function is used in conjunction with the **SPCSetMeasurement** function.

Syntax `SPCSaveSample("Dataset");`

Parameter	Description
Dataset	Actual dataset name. Actual string or message tagname.

Remarks Processing this function will force the sample to be entered into the specified Dataset name. It will use the current values of the manual input variables for the measurements in the sample. The input values are set either via the DDE tags **MI_Mx** (x =the measurement number form 1 to 25) or by using the **SPCSetMeasurement()** function (measurements 1 to 300).

See Also **SPCSetMeasurement()**

SPCSelectDataset() SPC

Shows a dialog from which the user may select a direct dataset.

Syntax `DatasetName=SPCSelectDataset()`

Remarks This script causes the **Select a Dataset** dialog box to open.

Once a Dataset name is selected, the function returns it to the *DatasetName* tagname. This selection can also be used to change the *DatasetName* of an Indirect Dataset.

SPCSelectProduct() SPC

Shows a dialog from which the user may select a product in a given dataset.

Syntax `ProductName=SPCSelectProduct(Dataset);`

Remarks Processing this script causes the **Select a Product** dialog box to open.

Once a Product name is selected, the function returns it to the *ProductName* tagname. This name can now be used to change the collected product in the *Dataset.*

SPCSetControlLimits() SPC

Allows convenient manual or event driven input of the control limit values for a Control Chart.

Syntax `SPCSetControlLimits("Dataset",XUCL,XLCL);`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
XUCL	Represents the value to be used for the UCL of the chart. Number or real tagname.
XLCL	Represents the value to be used for the LCL of the chart. Number or real tagname.

Remarks These measurements are saved to the sample by processing **SPCSaveSample()**.

See Also **SPCSaveSample(), SPCSetRangeLimits(), SPCSetSpecLimits()**

SPCSetMeasurement() SPC

Allows convenient manual or event driven inputs of analog measurement values by processing the script.

Syntax `SPCSetMeasurement("Dataset",Measurement,Value);`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
Measurement	Represents a measurement number (from 1 to 300). Number or integer tagname.
Value	The value to be written to the specified measurement number. Number or real tagname.

Remarks Use **SPCSaveSample()** once all measurements are set to save the data to the database.

SPCSetProductCollected() SPC

Changes the product being collected in a specified Dataset.

Syntax `SPCSetProductCollected("Dataset","Product");`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
Product	Contains the Product name under which data should be collected. Actual string or message tagname.

Remarks This function does not change the product being displayed. It is possible to collect data for one product and display data for another by using this function to collect and the **SPCSetProductDisplayed()** function to display.

Example
```
SPCSetProductCollected("Data5838", "Widgets");
```

See Also **SPCSetProductDisplayed()**

SPCSetProductDisplayed() SPC

Changes the product being displayed in a specified Dataset.

Syntax `SPCSetProductDisplayed("Dataset","Product");`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
Product	Contains the product name for which data will be displayed. Actual string or message tagname.

Example
```
SPCSetProductDisplayed("ADataSetName", "AProductName");
```

See Also **SPCSetProductCollected()**

SPCSetRangeLimits() SPC

Allows convenient manual or event driven input of the Control Limits for a Range Chart.

Syntax `SPCSetRangeLimits("Dataset",RUCL,RLCL);`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
RUCL	Represents the value to be used for the UCL for a Range Chart. Number or real tagname.
RLCL	Represents the value to be used for the LCL for a Range Chart. Number or real tagname.

See Also **SPCSetControlLimits(), SPCSetSpecLimits()**

SPCSetSpecLimits() SPC

Allows convenient manual or event driven input of the specification limit values for a Control Chart.

Syntax `SPCSetSpecLimits("Dataset",XUSL,XLSL);`

Parameters	Description
Dataset	Contains the actual Dataset name. Actual string or message tagname.
XUSL	Represents the value to be used for the chart's USL. Number or real tagname.
XLSL	Represents the value to be used for the chart's LSL. Number or real tagname.

See Also **SPCSetControlLimits(), SPCSetRangeLimits()**

SQLAppendStatement() SQL

Continues a SQL statement using the contents of *string*. Errors are returned in the function return.

Syntax `[ResultCode=]SQLAppendStatement(ConnectionID,"SQLStatement");`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
SQLStatement	Actual statement to append.

Example

```
[ResultCode=]SQLAppendStatement(ConnectionID,"where
tablename.columnname=(any value or string)");
```

See Also **SQLConnect()**

SQLClearParam() SQL

Clears the value of the specified parameter. SQLSetParam must be called again before calling the **SQLExecute()** function.

Syntax `[ResultCode=]SQLClearParam(SQLHandle,ParameterNumber);`

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	The actual parameter within the SQL statement to modify.

See Also **SQLPrepareStatement(), SQLExecute()**

SQLClearStatement() SQL

Releases the resources associated with the statement specified by *SQLHandle*.

Syntax `[ResultCode=]SQLClearStatement(ConnectionID,SQLHandle);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.

See Also **SQLConnect()**

SQLClearTable() SQL

Deletes all records in a database table, but keeps the table.

Syntax `[ResultCode=]SQLClearTable(ConnectionID,"TableName");`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	The name of the database table you want to access.

Example To delete all records in the table BATCH1, the following statement would be used:

```
[ResultCode=]SQLClearTable(ConnectionID,"BATCH1");
```

See Also **SQLConnect()**

SQLCommit() SQL

The **SQLCommit()** command defines the end of a group of transaction commands. The group of commands performed between the **SQLTransact()** command and the **SQLCommit()** command is called a transaction set. A transaction set is handled like a single transaction. After the **SQLTransact()** command is issued, all subsequent operations will not be committed to the database until the **SQLCommit()** command is issued.

Syntax

[ResultCode=]**SQLCommit**(*ConnectionID*);

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Note Use caution when writing scripts that include the **SQLCommit()** command. Since the processing time is multiplied by the number of commands in a transaction set, including numerous commands will significantly slow performance.

Example

```
ResultCode = SQLTransact( ConnectionID);
ResultCode = SQLInsertPrepare( ConnectionID, TableName, BindList,
SQLHandle );
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertEnd( ConnectionID, SQLHandle );
ResultCode = SQLCommit( ConnectionID);

{Database keeps the 3 Inserts }
```

See Also **SQLRollback(), SQLTransact()**

SQLConnect() SQL

Connects InTouch to the database specified in the *ConnectString*.

Syntax

```
[ResultCode=]SQLConnect(ConnectionID,"ConnectString");
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
ConnectString	String that identifies the database and any additional logon information used in **SQLConnect()** function.

Example

The following statements connects to IBM OS/2 Database Manager and to the database named SAMPLE:

```
[ResultCode=]SQLConnect(ConnectionID,"DSN=OS2DM;DB=SAMPLE");
```

This function returns a value to the *ConnectionID* variable that is used as a parameter in all subsequent SQL Functions.

The *ConnectString* identifies the database system and any additional logon information. It is entered in the following format:

```
"DSN=data source name[;attribute=value[;attribute=value]...]"
```

The attributes required by each database vary. QELIB recognizes the following attributes for all databases:

For more information on the attributes supported by a specific database, see Chapter 2 of the *SQL Access for InTouch User's Guide.*

Attribute	Value
DSN	The name of the data source as configured in the Microsoft ODBC Administrator.
DLG	When enabled (DLG=1), displays a dialog box that allows you to input of connection string information.
DRV	For compatibility with SQL Access for InTouch version 4.11, this value is used if a data source name (DSN) is not present in the connection string. QELIB changes it to the data source name.
UID	Logon ID.
PWD	Password.
MODIFYSQL	Used by QELIB to ensure compatibility between the SQL used in the application and the SQL used in the database. When set to 1 (default), the database driver expects ODBC-compliant syntax, which it will modify as necessary for the underlying database system. When set to 0, the database driver expects and supports the native syntax of the underlying database system. This enables you to continue using applications developed with the SQL supported by the QELIB 1.0 database drivers.
REREADAFTERUPDATE	If enabled (set to 1), QELIB rereads a record from the database after updating it. This is useful for getting the correct value of auto-updated columns such as time stamps.
REREADAFTERINSERT	If enabled (set to 1), QELIB rereads a record from the database after inserting it. This is useful for getting the correct value of auto-updated columns such as time stamps.

SQLCreateTable() SQL

Creates a table in the database using the parameters in the named Table Template. Table Templates (defined in the SQL.DEF file) determine the structure of a database table.

Syntax

```
[ResultCode=]SQLCreateTable(ConnectionID,TableName,TemplateName);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	Name of the database table you want to access.
TemplateName	Name of the template definition you want to use.

Example

The following statement creates a table named *BATCH1* with the column names and types defined in *TEMPLATE*:

```
[ResultCode=]SQLCreateTable(ConnectionID,"BATCH1","TEMPLATE");
```

Note When an parameter is entered in a script surrounded by quotation marks, for example, "Parameter1", that exact string will be used. If no quotation marks are used, Parameter1 is assumed to be a tagname and the system will access the InTouch tagname dictionary for the value of the tagname, Parameter1. For example:

```
"c:\main\file" vs. location
```

where: location is an InTouch message tagname

```
"c:\main\file" is a literal string
```

See Also **SQLConnect()**

SQLDelete() SQL

Deletes a record or multiple records.

Syntax

```
[ResultCode=]SQLDelete(ConnectionID,TableName,WhereExpr);
```

Note The **SQLDelete()** function can not contain a null *WhereExpression*.

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	The name of the database table to access.

WhereExpression	Defines a condition that can be either true or false for any row of the table. The function extracts only those rows from the table for which the condition is true. The expression must be in the following format:
	ColumnName *comparison_operator* expression.
	Note If the column is a character data type, the expression must be in single quotes.
	The following example will select all rows whose name column contains the value EmployeeID:
	`name='EmployeeID'`
	The following example will select all rows containing part numbers from 100 to 199:
	`partno>=100 and partno<200`
	The following example will select all rows whose temperature column contains a value that is greater than 350:
	`temperature>350`

Example

The following statement deletes all records in table *BATCH1* whose lot number is equal to 65:

```
[ResultCode=]SQLDelete(ConnectionID,"BATCH1","lotno=65");
```

See Also

SQLConnect()

SQLDisconnect() SQL

Disconnects the user from the database.

Syntax `[ResultCode=]SQLDisconnect(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

See Also **SQLConnect()**

SQLDropTable() SQL

Destroys a table.

Syntax `[ResultCode=]SQLDropTable(ConnectionID,TableName);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	Name of database table name you want to access.

Example The following statement destroys the table *BATCH1*, once this command is given, the table name will no longer recognized and will not respond to any commands:

```
[ResultCode=]SQLDropTable(ConnectionID,"BATCH1");
```

See Also **SQLConnect()**

SQLEnd() SQL

This is used after a **SQLSelect()** function to free resources that were being used to store the Results Table.

Syntax `[ResultCode=]SQLEnd(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

See Also **SQLConnect(), SQLSelect()**

SQLErrorMsg() SQL

Retrieves the text error message associated with a specific *ResultCode*. *ErrorMsg* is the InTouch memory message tagname (maximum 131 characters) associated with *ResultCode*.

Syntax `SQLErrorMsg(ResultCode);`

Parameter	Description
ResultCode	An integer variable returned from most SQL functions. It is returned as a 0 if the function was successful and a negative integer if it fails.

For more information on these codes, see Appendix A, "Troubleshooting SQL Script Functions".

Example

```
ErrorMsg=SQLErrorMsg(ResultCode);
```

See Also SQLConnect()

SQLExecute() SQL

Executes the SQL statement. If the statement is a select statement, the *BindList* parameter designates the name of the *BindList* to use for binding the database columns with tagnames. If the *BindList* is NULL, no tagnames relationships are formed. For example, the SQL statement could be Create View, Insert, etc. Errors are returned in the function return. If the statement has been "prepared," the statement handle returned from the prepare should be passed. If the statement has not been prepared, the statement handle should be zero (0).

Syntax `[ResultCode=]SQLExecute(ConnectionID,BindList,SQLHandle);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
BindList	Defines which InTouch tagnames are used and which database columns they are associated with.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.

See Also SQLConnect(),SQLPrepareStatement()

Note **SQLExecute()** function can only be called once for a statement that has not been prepared. If the statement has been prepared, it can be called numerous times.

SQLFirst() SQL

Selects the first record of the Results Table created by the last **SQLSelect()** function. A **SQLSelect()** function must be processed before using this command.

Syntax `[ResultCode=]SQLFirst(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

See Also **SQLConnect(), SQLSelect()**

SQLGetRecord() SQL

Retrieves the record specified by *RecordNumber* from the current selection buffer.

Syntax `[ResultCode=]SQLGetRecord(ConnectionID,Record Number);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
RecordNumber	Actual record number to retrieve.

Example
```
[ResultCode=]SQLGetRecord(ConnectionID,3);
```

See Also **SQLConnect()**

SQLInsert() SQL

Inserts a new record into the referenced table using the values of the tagnames in the supplied BindList. The *BindList* parameter defines which InTouch tagnames are used and which database columns they are associated.

Syntax

```
[ResultCode=]SQLInsert(ConnectionID,TableName,BindList);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	Name of the database table you want to access.
BindList	Defines which InTouch tagnames are used and which database columns they are associated with.

Example

The following statement inserts a new record into table ORG with the tagname values specified in List1:

```
[ResultCode=]SQLInsert(ConnectionID,"ORG","List1");
```

Note The following three functions can be used in place of the standard **SQLInsert()** function to quickly insert records into a file. The **SQLInsert()** function is a one step operation that inserts and ends the statement. Because of this, the next time a **SQLInsert()** function is called, another whole operation is done again. This takes much longer than using the three functions below. The three functions below break these steps apart, so that once you do a **SQLInsertPrepare()** function, you can do as many **SQLInsertExecute()** functions you want, as fast as you want, then do a **SQLInsertEnd()** when finished.

See Also **SQLConnect(), SQLInsert(), SQLInsertPrepare(), SQLInsertExecute(), SQLInsertEnd()**

SQLInsertEnd() SQL

Releases the statement.

Syntax

```
[ResultCode=]SQLInsertEnd(ConnectionID,SQLHandle);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used

See Also **SQLConnect(), SQLPrepareStatement()**

SQLInsertExecute() SQL

Execute the already prepared statement.

Syntax

```
[ResultCode=]SQLInsertExecute(ConnectionID,BindList,SQLHandle);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
BindList	Defines which InTouch tagnames are used and which database columns they are associated with.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.

See Also **SQLConnect(), SQLPrepareStatement()**

SQLInsertPrepare() SQL

Creates and prepares an Insert statement for execution. The Insert statement is not processed. The *SQLHandle* parameter is an integer tagname that will contain a value after the statement is processed.

Syntax

```
[ResultCode=]SQLInsertPrepare(ConnectionID,TableName,BindList,
SQLHandle);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	The name of the database table to access.
BindList	Defines which InTouch tagnames are used and which database columns they are associated with.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.

See Also **SQLConnect(), SQLPrepareStatement()**

SQLLast() SQL

Selects the last record of the Results Table created by the last **SQLSelect()** function. A **SQLSelect()** function must be processed before using this command.

Syntax

```
[ResultCode=]SQLLast(ConnectionID);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Example

```
[ResultCode=]SQLLast(ConnectionID);
```

See Also **SQLConnect(), SQLSelect()**

SQLLoadStatement() SQL

Reads the statement contained in *FileName*. At this point the statement is similar to one created by **SQLSetStatement()** function, and can be appended to via **SQLAppendStatement()** function, or executed by **SQLExecute()** function. There can be only one statement per file. However, **SQLAppendStatement()** function can be used to append something to the statement if **SQLPrepareStatement()** function or **SQLExecute()** function has not been called.

Syntax

```
[ResultCode=]SQLLoadStatement(ConnectionID,FileName);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
FileName	The name of the file name in which the information is contained.

Remarks Prepare the SQL Statement that has been created by either **SQLSetStatement**, or **SQLLoadStatement** functions. The statement handle is returned.

Example

```
[ResultCode=]SQLLoadStatement(ConnectionID,"C:\InTouchAppname\SQL
.txt")
SQL.txt = Select ColumnName from TableName where ColumnName>100;
```

See Also **SQLConnect(), SQLAppendStatement(), SQLExecute(), SQLPrepareStatement**

SQLManageDSN() SQL

Runs the Microsoft ODBC Manager setup program. This can be used to add, delete and modify all data source names.

Syntax `SQLManageDSN(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

SQLNext() SQL

Selects the next record of the Results Table created by the last **SQLSelect()** function. A **SQLSelect()** function must be processed before using this command.

Syntax `[ResultCode=]SQLNext(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Example

```
[ResultCode=]SQLNext(ConnectionID);
```

See Also **SQLConnect(), SQLSelect()**

SQLNumRows() SQL

Indicates how many rows met the criteria specified in the last **SQLSelect()** function. For example, if a *WhereExpression* is used to select all rows with a column name AGE, where AGE is equal to 45, the number of rows returned could be 40 or 4000. This may determine which function is processed next.

Syntax `SQLNumRows(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Example The following statement returns the number of rows selected to the integer tagname NumRows:

```
NumRows=SQLNumRows(ConnectionID);
```

See Also **SQLConnect()**

SQLPrepareStatement() SQL

A **SQLPrepareStatement()** prepares an existing SQL statement for use by the **SQLSetParam()** function. A statement can be created by using either a **SQLSetStatement()**, or **SQLLoadStatement()**. The statement handle is returned.

Syntax `[ResultCode=]SQLPrepareStatement(ConnectionID,SQLHandle);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.

Example `[ResultCode=]SQLPrepareStatement(ConnectionID,SQLHandle);`

See Also **SQLConnect(), SQLSelect(), SQLSetStatement(), SQLLoadStatement()**

SQLPrev() SQL

Selects the previous record of the Results Table created by the last **SQLSelect()** function.

Syntax `[ResultCode=]SQLPrev(ConnectionID);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Remarks A **SQLSelect()** function must be processed before using this command.

Example `[ResultCode=]SQLPrev(ConnectionID);`

See Also **SQLConnect(), SQLSelect()**

SQLRollback() SQL

The **SQLRollback()** command reverses, or "rolls back," the most recently committed transaction set. A transaction set is a group of commands issued between the **SQLTransact()** command and the **SQLCommit()** command or the **SQLRollback()** command. A transaction set is handled like a single transaction. After the **SQLTransact()** command is issued, all subsequent operations are not committed to the database until the **SQLCommit()** command is issued.

Syntax

`[ResultCode=]`**`SQLRollback`**`(`*`ConnectionID,`*`);`

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Example

```
ResultCode =SQLTransact( ConnectionID);
ResultCode = SQLInsertPrepare( ConnectionID, TableName, BindList,
SQLHandle );
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertExecute( ConnectionID, BindList, SQLHandle
);
ResultCode = SQLInsertEnd( ConnectionID, SQLHandle );
ResultCode =SQLRollback( ConnectionID);

{Ignores all commands after Transact()  Database unchanged }
```

See Also **SQLCommit(), SQLTransact()**

SQLSelect() SQL

Instructs the database to retrieve information from a table. When a **SQLSelect()** function is processed, a temporary Results Table is created in memory, containing records that can be browsed using **SQLFirst()**, **SQLLast()**, **SQLNext()** and **SQLPrev()** functions.

Note Always use the **SQLEnd()** function after a **SQLSelect()** to free resources that were being used to store the Results Table.

Syntax

```
[ResultCode=]SQLSelect(ConnectionID,TableName,BindList,
WhereExpr,OrderByExpr);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	The name of the database table to access.
BindList	Defines which InTouch tagnames are used and which database.

WhereExpression	Defines a condition that can be either true or false for any row of the table. The command extracts only those rows from the table for which the condition is true. The expression must be in the following format:
	ColumnName comparison_operator expression
	Note If the column is a character data type, the expression must be in single quotes.
	The following example will select all rows whose name column contains the value EmployeeID:
	`name='EmployeeID'`
	The following example will select all rows containing part numbers from 100 to 199:
	`partno>=100 and partno<200`
	The following example will select all rows whose temperature column contains a value that is greater than 350:
	`temperature>350`

WhereExpression Examples:

WhereExpr - Memory message Tag
OrderByExpr - Memory message Tag
Speed_Input - Memory Real - User Input Analog
Serial_Input - Memory Message - User Input String

Analog example

```
WhereExpr = "Speed = " + text(Speed_Input,"#.##");
```

Because Speed_Input is a number it must be converted to text so it can be concatenated to the where expression string.

String example

```
WhereExpr = "Ser_No = '" + Serial_input + "'";
```

Because Serial_Input is a string it must have single quotes around the value for example:WhereExpr = "Ser_No='125gh'";

String example using the like statement

```
WhereExpr = "Ser_No like '" + "125%'"
```

When using Like the % char can be used as a wild card.

String and Analog example using And

```
WhereExpr = “Ser_No = ‘“ + Serial_input + “’” + “ and “ + “Speed
= “ + text(Speed_Input,”#.##”);
OrderByExpr = “”;
```

If the order does not matter use a null string as shown above.

SQLSelect using WhereExpr tag

```
ResultCode =
SQLSelect(Connect_Id,TableName,BindList,WhereExpr,OrderByExpr);

Error_msg = SQLErrorMsg( ResultCode );
```

SQLSelect WhereExpr built in function

```
ResultCode =    SQLSelect(Connect_Id,TableName,BindList, “Ser_No
= ‘“ + Serial_input + “’”, OrderByExpr);

Error_msg = SQLErrorMsg( ResultCode );
```

Note Always call the **SQLEnd**(*Connect_Id*) function when the **SQLSelect()** has been completed. If the **SQLEnd()** function is not called the resources will not be released and your application will run out of memory.

OrderByExpression Defines the columns and direction for sorting. Only column names can be used to sort and the expression must be in this form:

ColumnName [ASC|DESC]

The following example will sort the selected table by the manager column in ascending order:

"manager ASC"

You can also sort by multi-columns where the expression is in the form:

ColumnName [ASC|DESC],

ColumnName [ASC|DESC]

The next example will sort the selected table by the temperature column in ascending order and the time column in descending order:

"temperature ASC, time DESC"

Example

The following statement records from the table BATCH using a BindList named List1, whose column name type contains cookie. It will present the information sorted by the amount column in ascending order and the sugar column in descending order:

```
[ResultCode=]SQLSelect(ConnectionID,"BATCH","List1","type='cookie
'"
,"amount ASC, sugar DESC");
```

The following statement selects all of the data in the database, do not specify a value for the *WhereExpression* and *OrderByExpression*:

```
[ResultCode=]SQLSelect(ConnectionID,"BATCH","List1","","");
```

See also

SQLFirst(), SQLConnect(), SQLLast(), SQLNext(), SQLPrev(), SQLEnd(), SQLSelect()

SQLSetParamChar() SQL

Sets the value of the specified parameter to the specified string. **SQLSetParamChar()** function may be call multiple times before executing, resulting in the parameter value being set to the concatenation of all values sent. Lengths of 0 (zero) are ignored.

Syntax

```
[ResultCode=]SQLSetParamChar(SQLHandle,ParameterNumber,
ParameterValue,MaxLen);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.
MaxLen	Maximum size of the column with which this parameter is associated. This setting determines whether the parameter is of varying character or long varying character type. If *MaxLen* is less than or equal to the largest character string allowed by the database, then the parameter is varying character type. If greater, long varying character type.

Example

```
[ResultCode=]SQLSetParamChar(SQLHandle,ParameterNumber,
ParameterValue,MaxLen);
```

See Also **SQLPrepareStatement()**

SQLSetParamDate() SQL

Sets the value of the specified date parameter to the specified string.

Syntax

```
[ResultCode=]SQLSetParamDate(SQLHandle,ParameterNumber,
ParameterValue);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.

Example

```
[ResultCode=]SQLSetParamDate(SQLHandle,ParameterNumber,
ParameterValue);
```

See Also **SQLPrepareStatement()**

SQLSetParamDateTime() SQL

Sets the value of the specified date-time parameter to the specified string.

Syntax

```
[ResultCode=]SQLSetParamDateTime(SQLHandle,ParameterNumber,
ParameterValue,Precision);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.
Precision	The length of the date-time value to be assigned. It is the number of characters in *ParameterValue* to use.

Example

```
[ResultCode=]SQLSetParamDateTime(SQLHandle,ParameterNumber,
ParameterValue,Precision);
```

See Also **SQLPrepareStatement()**

SQLSetParamDecimal() SQL

Sets the value of the specified decimal parameter to the specified string. *Precision* is the number of digits in the value and *Scale* is the number of digits to the right of the decimal point.

Syntax

```
[ResultCode=]SQLSetParamDecimal(SQLHandle,ParameterNumber,
ParameterValue,Precision,Scale);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.
Precision	The length of the date-time value to be assigned. It is the number of characters in *ParameterValue* to use.
Scale	Number of digits to the right of decimal point.

Example

```
[ResultCode=]SQLSetParamDecimal(SQLHandle,ParameterNumber,
ParameterValue,Precision, Scale);
```

See Also **SQLPrepareStatement()**

SQLSetParamFloat() SQL

Sets the value of the specified parameter to the specified *ParameterValue*.

Syntax

```
[ResultCode=]SQLSetParamFloat(SQLHandle,ParameterNumber,
ParameterValue);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.

Example

```
[ResultCode=]SQLSetParamFloat(SQLHandle,ParameterNumber,
ParameterValue);
```

See Also **SQLPrepareStatement()**

SQLSetParamInt() SQL

Sets the value of the specified parameter to the specified *ParameterValue*.

Syntax

```
[ResultCode=]SQLSetParamInt(SQLHandle,ParameterNumber,
ParameterValue);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.

Example

```
[ResultCode=]SQLSetParamInt(SQLHandle,ParameterNumber,
ParameterValue);
```

See Also **SQLPrepareStatement()**

SQLSetParamLong() SQL

Sets the value of the specified parameter to the specified *ParameterValue*.

Syntax

```
[ResultCode=]SQLSetParamLong(SQLHandle,ParameterNumber,
ParameterValue);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.

Example

```
[ResultCode=]SQLSetParamLong(SQLHandle,ParameterNumber,
ParameterValue);
```

See Also **SQLPrepareStatement()**

SQLSetParamNull() SQL

Sets the value of the specified parameter to the NULL.

Syntax

```
[ResultCode=]SQLSetParamNull(SQLHandle,ParameterNumber,
ParameterType,Precision,Scale);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterType	Data type of the specified parameter:

Type	Value	Description
Char	1	Blank Padded fixed length string
Var Char	2	Variable Length String
Decimal	3	BCD Number
Integer	4	4-byte Signed integer
Small integer	5	2-byte signed integer
Float	6	4-byte floating point
Double Precision Float	7	8-byte floating point
DateTime	8	26-byte date time value
Date	111	26-byte date time value
Time	112	26-byte date time value
No Type	0	No Data Type

Precision	Decimal value's precision, the max. size of the char., or the length in bytes of the date-time value.
Scale	Decimal value's scale. This value is required only if applicable to the parameter being set to null.

Remarks No type may be used if a call to **SQLSetParam** has already been done on this parameter.

Example

```
[ResultCode=]SQLSetParamNull(SQLHandle,ParameterNumber,
ParameterType,Precision,Scale);
```

See Also **SQLPrepareStatement()**

SQLSetParamTime() SQL

Sets the value of the specified time parameter to the specified string.

Syntax

```
[ResultCode=]SQLSetParamTime(SQLHandle,ParameterNumber,
ParameterValue);
```

Parameter	Description
SQLHandle	Integer value returned by SQL when a **SQLPrepareStatement()** function is used.
ParameterNumber	Actual parameter number in the statement.
ParameterValue	Actual value to set.

Example

```
[ResultCode=]SQLSetParamTime(SQLHandle,ParameterNumber,
ParameterValue);
```

See Also **SQLPrepareStatement()**

SQLSetStatement() SQL

Starts a SQL statement buffer using the contents of *SQLStatement*, on the established connection, *ConnectionID.* There can be one SQL Statement buffer per *ConnectionID*. Errors are returned in the function return.

Syntax

```
[ResultCode=]SQLSetStatement(ConnectionID,SQLStatement);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
SQLStatement	Actual statement, see the following example.

Example

```
[ResultCode=]SQLSetStatement(ConnectionID,"Select LotNo, LotName
from LotInfo");
```

In the example below the SQLHandle is set to zero so the statement does not have to call SQLPrepare(Connect_Id, SQLHandle) before the execute statement. Because the SQLhandle was not created by the SQLPrepare to properly end this select use the SQLEnd function instead of the SQLClearStatement().

```
SQLSetStatement( Connect_Id, "Select  Speed, Ser_No from
tablename where Ser_No ='" +   Serial_input  + "'");

SQLExceute(Connect_Id,0);
```

In the example below the SQLhandle is created by the SQLPrepareStatement and used in the SQLExceute function. To end this select statement use SQLClearStatement to free up resources and free the SQLhandle.

```
SQLSetStatement( Connect_Id, "Select  Speed, Ser_No from
tablename where Ser_No ='" +   Serial_input  + "'");
SQLPrepareStatement(Connect_Id,SQlHandle);

SQLExceute(Connect_Id,SqlHandle);
```

See Also **SQLConnect()**

SQLTransact() SQL

The **SQLTransact()** command defines the beginning of a group of transaction commands. The group of commands performed between the **SQLTransact()** command and the **SQLCommit()** command is called a transaction set. A transaction set is handled like a single transaction. After the **SQLTransact()** command is issued, all subsequent operations will not be committed to the database until the **SQLCommit()** command is issued.

Syntax

```
[ResultCode=]SQLTransact(ConnectionID);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Note Exercise caution when writing scripts that include the **SQLTransact()** command. Since the processing time is multiplied by the number of commands in a transaction set, including numerous commands will significantly slow performance.

See Also **SQLCommit(), SQLRollback()**

SQLUpdate() SQL

Modifies a record to update the record with the current tagname values.

Syntax

```
[ResultCode=]SQLUpdate(ConnectionID,TableName,BindList,
WhereExpr);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.
TableName	The name of the database table to access.
BindList	Defines which InTouch tagnames are used and which database columns they are associated with.

WhereExpression	Defines a condition that can be either true or false for any row of the table. The function extracts only those rows from the table for which the condition is true. The expression must be in the following format:
	ColumnName *comparison_operator* expression
	Note If the column is a character data type, the expression must be in single quotes.
	The following example will select all rows whose name column contains the value EmployeeID:
	`name='EmployeeID'`
	The following example will select all rows containing part numbers from 100 to 199:
	`partno>=100 and partno<200`
	The following example will select all rows whose temperature column contains a value that is greater than 350:
	`temperature>350`

Example

The following statement updates all records in the table BATCH, whose lot number is 65, to the current values of the tagnames specified in the BindList "List1":

```
[ResultCode=]SQLUpdate(ConnectionID,"BATCH","List1","lotno=65");
```

Note Be sure that all records are unique. If identical records exist in a table, both will be updated.

See Also

SQLConnect()

SQLUpdateCurrent() SQL

Takes the currently selected record and updates it with any new InTouch values. The following example will update the currently selected record.

Syntax

```
[ResultCode=]SQLUpdateCurrent(ConnectionID);
```

Parameter	Description
ConnectionID	Is a memory integer tagname created by the user to hold the number (ID) assigned by the **SQLConnect** function to each database connection.

Example

```
[ResultCode=]SQLUpdateCurrent(ConnectionID);
```

See Also **SQLConnect()**

Sqrt() math

Causes InTouch to automatically calculate the square root of the value which follows the statement.

Syntax

```
Real Result=Sqrt(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.
Real Result	A real type tagname used to store the result.

Example

```
AnalogTag1=Sqrt(AnalogTag2);
```

StartApp system

Automatically starts another Windows application.

Syntax

```
StartApp "AppName";
```

Parameter	Description
AppName	Actual program name of the application to be started, for example, *Wordpad.exe*.

It is recommended that the .EXE extension of the program name be entered. Command line parameters may also be entered, if the executable supports them.

Long file names do not work, however using the following example using the Dos equivalent for a long file name will work.

If the long file name is:C:\Program files\Microsoft Office\Office\Excel. Use C:\Progra~1\Micros~2\Office\Excel (Dos equivalent).

```
SetApp "C:\Progra~1\Micros~2\Office\Excel";
```

If Winfile is used and all file details are shown the center column in the window will show the Dos equivalent file path.

Examples

The following statement starts the Microsoft Windows Wordpad program.

```
StartApp "Wordpad.exe";
```

See Also

ActivateApp()

StringASCII() string

Returns the ASCII value of the first character in a specified message tagname.

Syntax

```
IntegerResult=StringASCII("Char");
```

Parameter	Description
Char	Alpha-numeric character or message tagname.

Remarks

The ASCII value of the first character in *Char* will be returned to *IntegerResult*. When this function is processed, only the single character is tested or affected. If the message tagname provided to *StringASCII* contains more than one character, only the first character of the tagname will be tested.

Examples

```
StringASCII("A") will return 65
StringASCII("A Mixer is Running") will return 65
StringASCII("a mixer is running") will return 97
```

See Also

StringChar()

StringChar() string

Returns the character corresponding to a specified ASCII code.

Syntax

```
MessageResult=StringChar(ASCII);
```

Parameter	Description
ASCII	ASCII code or integer tagname.

Remarks

The ASCII character specified by *ASCII* is returned to *MessageResult*. One use of this function may be to add ASCII characters not normally represented on the keyboard to message tags.

Example

```
ControlString=MessageTag+StringChar(13)+StringChar(10);
```

A [Carriage Return (13)] and [Line Feed (10)] have been added to the end of *MessageTag* and passed to *ControlString*. Inserting characters out of the normal 32-126 range of displayable ASCII characters can be very useful for creating control codes for external devices such as printers or modems.

A common use of this function is for SQL commands. The where expression sometimes requires double quotes around string values, so **StringChar**(34) is used.

See Also **StringASCII()**

StringFromIntg() string

Converts an integer value into its string representation in another base.

Syntax

```
MessageResult=StringFromIntg(Number,Base);
```

Parameter	Description
Number	Number to convert. Number or integer tagname.
Base	Base to use in conversion. Number or integer tagname.

Remarks

Integer is converted to the *Base* specified and the result is stored in *MessageResult*.

Examples

```
StringFromIntg(26, 2) will return "11010"
StringFromIntg(26, 8) will return "32"
StringFromIntg(26, 16) will return "1A"
```

See Also **StringToIntg(), StringFromReal(), StringToReal(), Text()**

StringFromReal() string

Converts a real value into its string representation, either as a floating-point number or in exponential notation.

Syntax

```
MessageResult=StringFromReal(Number,Precision,"Type");
```

Parameters	Description
Number	Is converted to the *Precision* and *Type* specified and the result is stored in *Message Result*. Number or real tagname.
Precision	Specifies how many decimal places will be shown. Number or integer tagname.
Type	Determines the display method:

Type	Description
"f"	Display in floating-point notation.
"e"	Display in exponential notation with a lower-case "e."
"E"	Display in exponential notation with an upper-case "E."

Examples

```
StringFromReal(263.355, 2,"f") will return "263.36"
StringFromReal(263.355, 2,"e") will return "2.63e2"
StringFromReal(263.55, 3,"E") will return "2.636E2"
```

See Also StringToReal(), StringFromIntg(), StringToIntg(), Text()

StringFromTime() string

Converts a time value (in seconds since Jan-01-1970) into a particular string representation.

Syntax

```
MessageResult=StringFromTime(SecsSince1-1-70,StringType);
```

Parameters	Description
SecsSince1-1-70	Is converted to the *StringType* specified and the result is stored in *MessageResult*.
StringType	Determines the display method:

Type	Description
1	Displays the date in the same format set from the windows control Panel. (Similar to that displayed for $DateString.)
2	Displays the time in the same format set from the windows control Panel. (Similar to that displayed for $TimeString.)
3	Displays a 24-character string indicating both the date and time: "Wed Jan 02 02:03:55 1993"
4	Displays the short form for the day of the week: "Wed"
5	Displays the long form for the day of the week: "Wednesday"

Examples

```
StringFromTime(86400, 1) will return "1/2/70"
StringFromTime(86400, 2) will return "12:00:00 AM"
StringFromTime(86400, 3) will return "Fri Jan 02 00:00:00 1970"
StringFromTime(86400, 4) will return "Fri"
StringFromTime(86400, 5) will return "Friday"
```

StringInString() string

Returns the position in *Text* where *SearchFor* first occurs.

Syntax

```
IntegerResult=StringInString("Text","SearchFor",StartPos,CaseSens);
```

Parameters	Description
Text	Is searched for occurrences of *SearchFor*. If multiple occurrences of *SearchFor* are found, the location of the first one will be returned to *IntegerTag*. Actual string or message tagname.
SearchFor	The sting to be searched for in Text. Actual string or message tagname.
StartPos	Is an integer that determines where in *Text* the search will begin. Number or integer tagname.
CaseSens	Parameter determines whether the search will be case sensitive (0=no and 1=yes). Number or integer tagname.

Example

```
StringInString("The mixer is running", "mix", 1, 0) will return 5
StringInString("Today is Thursday", "day", 1, 0) will return 3
StringInString("Today is Thursday", "day", 10, 0) will return 15
StringInString("Today is Veteran's Day", "Day", 1, 1) will return 20
StringInString("Today is Veteran's Day", "Night", 1, 1) will return 0
```

StringLeft() string

Returns the number of characters specified by Chars starting with the leftmost character of text.

Syntax

```
MessageResult=StringLeft("Text",Chars);
```

Parameters	Description
Text	Actual string or message tagname
Chars	Number of characters to return or integer tagname.

Remarks

If *Chars* is set to 0, the entire string will be returned.

Example

```
StringLeft("The Control Pump is On", 3) will return "The"
StringLeft("Pump 01 is On", 4) will return "Pump"
StringLeft("Pump 01 is On", 96) will return "Pump 01 is On"
StringLeft("The Control Pump is On", 0) will return "The Control Pump is On"
```

StringLen() string

Returns the length of text to integer result.

Syntax

```
IntegerResult=StringLen("Text");
```

Parameters	Description
Text	Actual string or a message tagname

Remarks The length (in characters) of *Text* returned to *IntegerTag*. All the characters in the message tagname, including those not normally displayed on the screen are counted.

Example

```
StringLen("Twelve percent") will return 14
StringLen("12%") will return 3
StringLen("The end." + StringChar(13)) will return 10
```

Note The carriage return character is ASCII 13

StringLower() string

Converts all the uppercase characters in text to lower case, and places the resulting sting in MessageResult.

Syntax

```
MessageResult=StringLower("Text");
```

Parameters	Description
Text	String to be converted to lower case. Actual string or a message tagname.

Remarks Lower case characters, symbols, numbers and other special characters will not be affected.

Examples

```
StringLower("TURBINE") will return "turbine"
StringLower("22.2 Is The Value") will return "22.2 is the value."
```

StringMid() string

Extract from text the specific numbers of characters specified by Chars, starting at the position *StartChar*. This function is slightly different from its counterparts **StringLeft()** function and **StringRight()** function in that it allows the user to specify both the start and end of the string which is to be extracted from the message tagname.

Syntax

```
MessageResult=StringMid("Text",StartChar,Chars);
```

Parameters	Description
Text	Actual string or a message tagname
StartChar	Specifies the position of the first character to extract. Number or integer tagname.
Chars	Specifies the total number of characters to return. Number or integer tagname.

Examples

```
StringMid("The Furnace is Overheating",5,7,) will return
 "Furnace"
StringMid("The Furnace is Overheating",13,3) will return "is "
StringMid("The Furnace is Overheating",16,50) will return
"Overheating"
```

See Also StringLeft(), StringRight()

StringReplace() string

Replaces or changes specific parts of a provided string. Using this function can take a string tagname and replace characters, words or phrases.

Syntax

```
MessageResult=StringReplace(Text,SearchFor,ReplaceWith,
CaseSens,NumToReplace,MatchWholeWords);
```

Parameters	Description
Text	The string that you wish to change. Actual string or message tagname.
SearchFor	The string you wish to search for and replace. Actual string or message tagname.
ReplaceWith	The replacement string. Actual string or message tagname.
CaseSens	Determines whether the search will be case sensitive. (0=no and 1=yes) Number or integer tagname.
NumToReplace	Determines the number of occurrences you wish to replace. (-1=all) Number or integer tagname.
MatchWholeWords	Determines whether the function will limit its replacement to whole words. (0=no and 1=yes) Number or integer tagname. If *MatchWholeWords* is turned on (set to 1) and the *SearchFor* is set to "and", the "and" in "handle" would not be replaced. If the *MatchWholeWords* is turned off (set to 0), it would be replaced.

Examples

```
StringReplace("In From Within","In","Out",0,1,0) returns "Out
From Within"  (replaces only the first one)

StringReplace("In From Within","In","Out",0,-1,0) returns "Out
From without"  (replaces all occurrences)

StringReplace("In From Within","In","Out",1,-1,0) returns "Out
From Within"  (replaces all that match case)

StringReplace("In From Within","In","Out",0,-1,1) returns "Out
From Within"  (replaces all that are whole words)
```

Note The **StringReplace()** function does not recognize special characters, for example, @#$%&*(). It sees them as delimiters. For example, if the function **StringReplace()** (abc#,abc#,1234,0,1,1) is processed, there will be no replacement. The # sign is recognized as a delimiter instead of a character.

StringRight() string

Returns the number of character specified by Chars starting at the rightmost character of text.

Syntax

```
MessageResult=StringRight("Text",Chars);
```

Parameters	Description
Text	Actual string or message tagname.
Chars	The number of characters to return or integer tagname.

Remarks

If *Chars* is set to 0, the entire string will be returned.

Examples

```
StringRight("The Pump is On", 2) will return "On"
StringRight("The Pump is On", 5) will return "is On"
StringRight("The Pump is On", 87) will return "The Pump is On"
StringRight("The Pump is On", 0) will return "The Pump is On"
```

StringSpace() string

Generates a string of spaces either within a message tagname or an expression.

Syntax

```
MessageResult=StringSpace(NumSpaces);
```

Parameters	Description
NumSpaces	Number of spaces to return. Number or integer tagname.

Remarks

StringSpace() function returns a string of spaces of a length specified by *NumSpaces*.

Examples

All spaces are represented by the "×" character:

```
StringSpace(4) will return "××××"
"Pump" + StringSpace(1) + "Station" will return "Pump×Station"
```

StringTest() string

Tests the first character of text to determine whether it is of a certain type.

Syntax

```
DiscreteResult=StringTest("Text",TestType)
```

Parameters	Description
Text	String that function will act on. Actual string or message tagname.
TestType	Determines one of the following:

Type	Description
1	Alphanumeric character ('A'-'Z', 'a-z' and '0-9')
2	Numeric character ('0'- 9')
3	Alphabetic character ('A-Z' and 'a-z')
4	Upper-case character ('A'-'Z')
5	Lower-case character ('a'-'z')
6	Punctuation character (0x21-0x2F)
7	ASCII characters (0x00 - 0x7F)
8	Hexadecimal characters ('A'-'F' or 'a'-'f' or '0'-'9')
9	Printable character (0x20-0x7E)
10	Control character (0x00-0x1F or 0x7F)
11	White Space characters (0x09-0x0D or 0x20)

Remarks

StringTest() function will return a positive value to *DiscreteResult* if the first character in *Text* is of the type specified by *TestType*. If the **StringTest()** function contains more than one character, only the first character of the tagname will be tested.

Examples

```
StringTest("ACB123",1) will return 1
StringTest("ABC123",5) will return 0
```

StringToIntg() string

Converts the numeric value of a message tagname to an integer value to which mathematical calculations can be applied.

Syntax

```
IntegerResult=StringToIntg("Text");
```

Parameters	Description
Text	String that function will act on. Actual string or message tagname.

Remarks

When this statement is evaluated, the system reads the first character of the string for a numeric value. If the first character is other than a number (blank spaces are ignored), the string's value is equated to zero (0). If the first character is found to be a number, the system continues to read the subsequent characters until a non-numeric value is seen.

Examples

```
If Text="ABCD", then IntegerTag=0.
If Text="22.2 is the Value", then IntegerTag=22
(since integers are whole numbers).
If Text="The Value is 22", then IntegerTag=0.
```

See Also

StringFromIntg(), StringFromReal(), StringToReal(), Text()

StringToReal() string

Converts the numeric value of a message tagname to a real (floating point) value to which mathematical calculations can be applied.

Syntax

```
RealResult=StringToReal("Text");
```

Parameters	Description
Text	String that function will act on. Actual string or message tagname.

Remarks

When this statement is evaluated, the system reads the first character of the string for a numeric value. If the first character is other than a number (blank spaces are ignored), the string's value is equated to zero (0). If the first character is found to be a number, the system continues to read the subsequent characters until a non-numeric value is seen.

Examples

```
If Text="ABCD", then RealTag=0.
If Text="22.261 is the Value", then RealTag=22.261.
If Text="The Value is 22", then RealTag=0.
```

See Also

StringFromReal(), StingFromIntg(), StringToIntg(), Text()

StringTrim() string

Removes unwanted spaces from text.

Syntax

```
MessageResult=StringTrim("Text",TrimType);
```

Parameters	Description
Text	String that will be trimmed of spaces. Actual string or message tagname.
TrimType	Determines one of the following:

Type	Description
1	Remove leading spaces (left of the first non-space character)
2	Remove trailing spaces (right of the last non-space character)
3	Remove all spaces except for single spaces between words

Remarks

Text is searched for white-spaces (ASCII 0x09-0x0D or 0x20) which are to be removed. *TrimType* determines the method used by the function:

Example

All spaces are represented by the "×" character.

```
StringTrim("×××××This×is×a××test×××××", 1) will return
"This×is×a××test×××××"
StringTrim("×××××This×is×a××test×××××", 2) will return
"×××××This×is×a××test"
StringTrim("×××××This×is×a××test×××××", 3) will return
"This×is×a×test"
```

Note The **StringReplace()** function can be used to remove ALL spaces from a specified message tagname. Simply replace all the space characters with a "null."

StringUpper() string

Converts all the lowercase characters in text to uppercase.

Syntax

```
MessageResult=StringUpper("Text");
```

Parameters	Description
Text	String to be converted to uppercase. Actual string or message tagname.

Remarks

Upper case characters, symbols, numbers and other special characters will not be affected.

Examples

```
StringUpper("abcd") will return "ABCD."
StringUpper("22.2 is the value") will return "22.2 IS THE VALUE"
```

Tan() math

Returns the *tangent* of an angle given in degrees.

Syntax

```
Result=Tan(AngleNumber);
```

Parameter	Description
AngleNumber	The angle in degrees. Any number, real or integer tagname.

Examples

```
Wave = 10 + 50 * Tan(6 * $seconds);
Tan(45) will return 1
Tan(0) will return 0
```

See Also

Sin(), Cos()

Text() string

Causes a message type tagname to display the value of an analog (integer or real) tagname based upon the specified *Format_Text*.

Syntax

```
MessageResult=Text(Analog_Tag,"Format_Text");
```

Parameters	Description
Analog_Tag	Number, real or integer tagname to convert.
Format_Text	Format to use in conversion. Actual string or message tagname.

Examples

```
MessageTag=Text(66,"#.00");
```

MessageTag is a message type tagname, *66* is either an integer or real type tagname and "*#0.00*" represents the display format for the equated value:

```
If Analog_Tag=66, then MessageTag=66.00.
```

```
If Analog_Tag=22.269, then MessageTag=22.27.
```

```
If Analog_Tag=9.999, then MessageTag=10.00.
```

```
LogMessage("The current value of FreeserRoomTemp is:" + Text
(FreezerRoomTemp, "#.#"));
```

In the following example, MessageTag will be set to "One=1 Two=2".

```
MessageTag = "One + " + Text(1,"#") + StringChar(32) + "Two +" +
Text(2,"#");
```

See Also StringFromIntg(), StringToIntg(), StringFromReal(), StringToReal()

Trunc() math

Truncates a real (floating point) number by simply eliminating the portion to the right of the decimal point.

Syntax

```
ResultNumericTag=Trunc(Number);
```

Parameter	Description
Number	Any number, real or integer tagname.

Remarks This function will accomplish the same result as placing the contents of a real tagname into an integer tagname.

Examples

```
Trunc(4.3) will return 4
```

```
Trunc(-4.3) will return -4
```

See Also Round()

wcAddItem() windows control

Adds the supplied string to the List box or Combo box. If the List box or Combo box was not created as sorted, the string is added to the end of the list. Otherwise the string is inserted into the list and the list is sorted.

Syntax

```
[ErrorNumber=]wcAddItem("ControlName","MessageTag");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
MessageTag	The message string to be displayed. Actual string or message tagname.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Examples

The following statement adds the contents of the message string to the List box when the window (using On Show Window script) containing the List box is opened:

```
wcAddItem("ListBox_1", "Chocolate");
wcAddItem("ListBox_1", "Vanilla");
wcAddItem("ListBox_1", "Strawberry");
```

See Also

wcInsertItem()

wcClear() windows control

Removes all items from the List box or Combo box.

Syntax

```
[ErrorNumber=]wcClear("ControlName");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement clears all items in a Combo box when a pushbutton (Action script) is pressed:

```
wcClear("ListBox_1");
```

wcDeleteItem() windows control

Deletes the item associated with the item index argument in both List or Combo boxes.

Syntax

```
[ErrorNumber=]wcDeleteItem("ControlName",ItemIndex);
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
ItemIndex	A number corresponding to the position of the item. Number or integer tagname.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement deletes the 3rd item in a list when a pushbutton (Action script) is pressed:

```
wcDeleteItem("ListBox_1", 3);
```

wcDeleteSelection() windows control

Deletes the currently selected item from the list.

Syntax

```
[ErrorNumber =]wcDeleteSelection("ControlName");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.

☞ For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement deletes the currently selected item in a Combo box list when a pushbutton (Action script) is pressed:

```
wcDeleteSelection("ListBox_1");
```

wcErrorMessage() windows control

Returns a message string describing the error.

Syntax

```
ErrorMessage=wcErrorMessage(ErrorNumber);
```

Parameter	Description
ErrorMessage	Message tagname.
ErrorNumber	Number returned by all windows control functions. Number or integer tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes, Text boxes, Combo boxes, Check boxes and Radio buttons.

Examples

If an error occurs while a list is being loaded, display the text description of the error into the message tagname *ErrorDescription*. In this example a String Value Output animation link was assigned to the tagname *ErrorDescription* to display the error message.

On Show Window Script:

```
ErrorNumber=wcLoadList("ListBox_1","c:\InTouch\recipe.txt");
ErrorDescription=wcErrorMessage(errornumber);
```

This function can also be used with all windows control functions to display error messages:

```
ErrorNumber=wcAddItem("ListBox_1","United States");
ErrorMsg=wcErrorMessage(ErrorNumber);
```

wcFindItem() windows control

Determines the corresponding index of the first item in the List box or Combo box that matches the supplied *Message* string.

Syntax

`[ErrorNumber=]`**`wcFindItem`** `(`*`"ControlName","MessageTag",DiscreteTag,`* *Tagname*);

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
MessageTag	The message string to be compared. Actual string or message tagname.
DiscreteTag	Determines the type of the string comparison. One of the following discrete values can be assigned: 0=case-insensitive 1=case-sensitive
Tagname	Actual name of an integer tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

wcGetItem() windows control

Returns the value property of the item string associated with the corresponding *ItemIndex* in the List box or Combo box.

Syntax

```
[ErrorNumber=]wcGetItem("ControlName",ItemIndex,Tagname);
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
ItemIndex	A number corresponding to the position of the item. Number or integer tagname.
Tagname	Actual name of a real or integer tagname. The **wcGetItem** function will place the numeric value corresponding to the item into this tagname upon return from the function.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement returns the string value of the 10th item in a Combo box to the message tagname *ListSelection* when a pushbutton (Action script) is pressed:

```
wcGetItem("Combobox_1", 10, ListSelection);
```

If item 10 in the list is "Vanilla" then *ListSelection* will contain the string Vanilla.

wcGetItemData() windows control

Determines the integer value associated with the list item identified by the parameter *ItemIndex*.

Syntax

```
[ErrorNumber=]wcGetItemData("ControlName",ItemIndex,Tagname);
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
ItemIndex	A number corresponding to the position of the item. Number or integer tagname.
Tagname	Actual name of a real or integer tagname. The **wcGetItemData** function will place the numeric value corresponding to the item into this tagname upon return from the function.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement retrieves the numeric value associated with the 5th item in a List box and returns it to the integer tagname *ItemValue* when a pushbutton (Action script) is pressed:

```
wcGetItemData("ListBox_1", 5, ItemValue);
```

If the 5th item in the list is assigned the integer value 4500 the tagname *ItemValue* will contain 4500.

See Also

wcSetItemData()

wcInsertItem() windows control

Inserts the *Message* string into the list. *ItemIndex* is the corresponding index number of the position in the List box that will receive the string. Unlike the **wcAddItem()**and **wcInsertItem()** functions will not sort a list, even if it was created as a sorted List box or Combo box.

Syntax

```
[ErrorNumber=]wcInsertItem("ControlName",ItemIndex,"MessageTag");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
ItemIndex	A number corresponding to the position of the item to be added. If this parameter is -1, the string is added to the end of the list. Number or integer tagname.
MessageTag	Contains the string to insert at the position indicated by *ItemIndex*. Actual string or message tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To List boxes and Combo boxes.

Example The following statement inserts a new item called "Blueberry" into a List box at 4th location from the top when a pushbutton (Action script) is pressed.

```
wcInsertItem("ListBox_1", 4, "Blueberry");
```

See Also **wcAddItem()**

wcLoadList() windows control

Replaces the contents of the List box or Combo box with the items contained in the *FileName*.

Syntax

```
[ErrorNumber=]wcLoadList("ControlName","Filename");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
Filename	Contains the name of a file in pure ASCII format. If a complete path name is not supplied as part of the message parameter, the function will check the application directory for the message file. Actual string or message tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To List boxes and Combo boxes.

Example The following statement loads a properly formatted list (located in c:\InTouch.32\wclist.txt.) into a Combo box when the window (On Show Window script) containing the Combo box is opened:

```
wcLoadList("Combobox_1", "c:\InTouch.32\wclist.txt");
```

See Also **wcAddItem(), wcSaveList()**

wcLoadText() windows control

Replaces the contents of the Text box with the contents of the *FileName*.

Syntax

```
[ErrorNumber=]wcLoadText("ControlName", "Filename");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
Filename	Contains the name of a file in pure ASCII format. If a complete path name is not supplied as part of the message parameter, the function will check the application directory for the message file. Actual string or message tagname.

Remarks

The **wcLoadText()** function only supports pure ASCII files such as those created using Microsoft's Notepad program. The Error Numbers for windows control are listed in Appendix A.

📖 For advanced file viewing capabilities see the document viewer from the Factory Suite's *Productivity Pack.*

Applies To

Text boxes.

Example

The following statement loads a Notepad text file (c:\InTouch.32\readme.txt.) into a Text box when the window (On Show Window script) containing the Text box is opened:

```
wcLoadText("Textbox_1", "c:\InTouch.32\readme.txt");
```

wcSaveList() windows control

Replaces the contents of the *FileName* with the items listed in the List box or Combo box object.

Syntax

```
[ErrorNumber=]wcSaveList("ControlName","Filename");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
Filename	Contains the name of a file in pure ASCII format. If the file does not exist, it is created. The items can subsequently be loaded into the list object with the **wcLoadList()** function. Actual string or message tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

List boxes and Combo boxes.

Example

The following statement saves the current items in a List box in a file **(c:\InTouch.32\newlist.txt)** when a pushbutton (Action script) is pressed:

```
wcSaveList("ListBox_1", "c:\InTouch.32\newlist.txt");
```

Note If external ASCII files are used to fill the List and Combo boxes they must follow specific formatting and contain specific information. Format:

ControlType, ListCount
ListItem, ItemData
ListItem, ItemData
: :
: :
ListItem, ItemData

For example: A list file is to be loaded to a Combo box and it contains three items to choose from and those items have no item data assigned (refer to the function **wcSetItemData()** function for more information on item data). The format of the file would appear as:

COMBOBOX, 3
Chocolate, 0
Vanilla, 0
Strawberry, 0

Description: COMBOBOX is the control type. The ListCount is 3 for Chocolate, Vanilla, and Strawberry it's 3 items. Chocolate is then listed as the first ListItem or index 1. Vanilla as index 2, and finally Strawberry as 3. Each of these index items has a data value of 0.

See Also

wcLoadList(), wcSetItemData()

wcSaveText() windows control

Saves the text contained in the Text box to *FileName*. If the files doesn't exist, it will be created. If it does exist it must be read/write.

Syntax

```
[ErrorNumber=]wcSaveText("ControlName","Filename");
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
Filename	Contains the name of a file in pure ASCII format. If a complete path name is not supplied the function will be saved in the application directory. If the file does not exist, it is created. The items can subsequently be loaded into the list object with the **wcLoadList()** function. Actual string or message tagname.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Applies To

Text boxes.

Example

The following statement saves the current information entered in a Text box to a file in c:\InTouch.32\newtext.txt when a pushbutton (Action script) is pressed:

```
wcSaveText("Textbox_1", "c:\InTouch.32\newtext.txt");
```

Note The **wcSaveText()** function only saves in a pure ASCII file such as those created using Microsoft's Notepad program.

See Also

wcLoadList()

wcSetItemData() windows control

Assigns an integer value of the item (*Number*) to the item in the list specified by *ItemIndex*. This function allows the assignment of a number to a string.

Syntax

```
[ErrorNumber=]wcSetItemData("ControlName",ItemIndex,Number);
```

Parameter	Description
ControlName	The name of the windows control object. for example, *ListBox_1*. Actual string or message tagname.
ItemIndex	An integer value specifying the order the requested items are to be displayed. Number or integer tagname.
Number	An integer value representing the item data. Number or integer tagname.

Remarks

Complete lists containing the items can be created externally (i.e. notepad) and then loaded using one function call. The list must be properly formatted as shown in the description of the **wcSaveList()** function.

For more information on error numbers, see Appendix A, "Error Messages for Windows Controls and Distributed Alarms".

Examples

There is a recipe that has three ingredients; flour, sugar and salt. The quantity of flour is 4500 grams, sugar is 1500 and salt is 325 grams. The values are assigned to each of the List box items by using a Data Change script triggered by what Recipe (tagname, RecipeName) is selected:

```
wcSetItemData("ListBox_1", 1, 4500); {set 1st item in the list (flour)=4500}
wcSetItemData("ListBox_1", 2, 1500); {set 2nd item in the list (sugar)=1500}
wcSetItemData("ListBox_1", 3, 325);   {set 3rd item in the list (salt)=325}
```

The **wcGetItemData()** function is used to return the value (item data) associated with the list item. The parameter *Tagname* contains the returned numeric value. This parameter could be a DDE integer tagname that writes directly to the real world device.

See Also

wcLoadList(), wcSaveList(), wcGetItemData()

WWControl() misc

Allows you to Restore, Minimize, Maximize, or Close an application from InTouch.

Syntax

```
WWControl("AppTitle","ControlType");
```

Parameter	Description
AppTitle	The name of the application title to be controlled. The application title for a particular application can also be determined using the **InfoAppTitle()** function. Actual string or message tagname.
ControlType	Determines how the application is controlled as follows (these actions are identical to clicking on their corresponding selections in the application's Control Menu). Actual string or message tagname.

Type	Description
"Restore"	Activates and displays the application's window.
"Minimize"	Activates a window and displays it as an icon.
"Maximize"	Activates and displays the application's window.
"Close"	Closes an application.

Examples

```
WWControl("Calculator","Restore");
WWControl("Microsoft Excel","Close");
WWControl(InfoAppTitle("View"), "Close");
```

See Also InFoAppTitle(), ActivateApp(), StartApp()

WWExecute() WWDDE

Executes a command (using a the DDE protocol) to a specified *Application* and *Topic*.

Syntax

```
[Status=]WWExecute("Application","Topic","Command");
```

Parameter	Description
Application	Application to send execute command to. Actual string or message tagname.
Topic	Topic within application to send execute command to. Actual string or message tagname.
Command	Command to send. Actual string or message tagname.

Remarks

The *Command* string will be sent to the particular *Application* and *Topic* specified.

Examples

The following statement executes a macro in Excel:

```
Macro="Macro1!TestMacro";
Command="[Run(" + StringChar(34) + Macro + StringChar(34) +
",0)]";
WWExecute("excel","system",Command);
```

When `WWExecute("excel","system",Command);` is processed, the following will be sent to Excel (and *TestMacro* will run):

```
[Run("Macro1!TestMacro")]
```

The following script executes a macro in Microsoft Access:

```
WWExecute("MSAccess","system","MyMacro");
```

The **WWExecute** function returns 1 if the application is running, the topic exists, and the command was sent successfully. It returns 0 when the application is busy and **-1** when there is an error. Therefore, the status of this command can be monitored:

```
Status=WWExecute("excel","system",Command);
```

Status is an integer tagname to which the 1, -1 or 0 will be written.

WWPoke() WWDDE

Pokes a value (using a DDE protocol) to a specified *Application*, *Topic*, and *Item*.

Syntax

```
[Status=]WWPoke("Application","Topic","Item","TextValue");
```

Parameter	Description
Application	Application to send Poke command to. Actual string or message tagname.
Topic	Topic within application to send Poke command. Actual string or message tagname.
Item	Item within topic to Poke. Actual string or message tagname.
TextValue	A message variable or string. If the value you wish to send is a number, you can convert it using the **Text()**, **StringFromIntg()**, or **StringFromReal()** functions. Actual string or message tagname.

Remarks

In this statement, the value *TextValue* will be sent to the particular *Application*, *Topic*, and *Item* specified.

Example

The following statement converts a value to text and will poke to an Excel spreadsheet cell:

```
String=Text(Value,"0");
WWPoke("excel","sheet1","r1c1",String);
```

📖 For more information on using Excel 5.0 with InTouch (DDE) see your *InTouch User's Guide*.

Note The behavior for **WWPoke()** from within the Application "View" to "View" is undefined and is not supported. The WWPoke() command is not guaranteed to succeed in this instance, and the command will probably time-out without the desired results.

The **WWPoke()** function returns 1 if the application is running, the topic and item exist, and the value was sent successfully. It returns 0 when the application is busy and **-1** when there is an error. Therefore, the status of this command can be monitored:

```
Status=WWPoke("excel","sheet1","r1c1",String);
```

Status is an integer tagname to which the 1, -1 or 0 will be written.

See Also

Text(), StringFromIntg(), StringFromReal()

WWRequest() WWDDE

Makes a one-time request for a value (using a DDE protocol) from a particular *Application*, *Topic*, and *Item*.

Syntax **WWRequest**(*Application,Topic,Item,ValueMsg_Tag*);

Parameter	Description
Application	Application to Request data from. Actual string or message tagname.
Topic	Topic within application to request data from. Actual string or message tagname.
Item	Item within topic to request data from. Actual string or message tagname.
ValueMsg_Tag	A message tagname enclosed in quotes which will contain the requested value from the *Application*, *Topic*, and *Item*. Actual string or message tagname.

Remarks In this statement, the DDE value in the particular *Application*, *Topic*, and *Item* will be returned into *ValueMsg_Tag*.

The value will be returned *as a string* into a Message Tag. If the value is a number, you can then convert it using the **StringToIntg()** or **StringToReal()** functions.

Examples The following statement requests a value from an Excel spreadsheet cell and convert the resulting string into a value:

```
WWRequest("excel","[Book1.xls]sheet1","r1c1",Result);
Value=StringToReal(Result);
```

The **WWRequest()** function returns 1 if the application is running, the topic and item exist, and the value was returned successfully. It returns 0 when the application is busy and -1 when there is an error. Therefore, the status of this command can be monitored:

```
Status=WWRequest("excel","[Book1.xls]sheet1","r1c1",Result);
```

Status is an integer tagname to which the 1, -1 or 0 will be written.

See Also **StringToIntg(), StringToReal()**

APPENDIX A

Troubleshooting QuickScript Functions

Several InTouch QuickScript functions return a value based on the status of their processing. This value, known as the result code or error code, can be used to determine the relative success of that function's processing. This appendix describes the result codes for Windows Controls, Recipe, and SQL script functions. Also described are the SPC DDE Items available for obtaining Dataset information and for controlling chart operations.

Error Messages for Windows Controls and Distributed Alarms

The Window Controls and Distributed Alarm script functions return a value based on the result of processing the script function. The return value, used for error diagnostics, can be assigned to an Integer tagname. For example:

```
ErrorNumber = wcGetItem("ControlName", Number, Tagname);
```

Where:

ErrorNumber is an **Integer** tagname that will hold the returned error value. The return value of the function can be passed to the **wcErrorMessage()**. The **wcErrorMessage()** will return a string description of the error. For example:

```
ErrorMsg = wcErrorMessge(ErrorNumber);
```

Where:

ErrorMsg is a Message type tagname that holds the text description of the returned error. The following table identifies the numeric value and its description:

Error Message	Description
0	Success
-1	General failure
-2	Insufficient memory available
-3	Property is read-only
-4	Specified item already present
-5	Object name unknown
-6	Property name unknown
-*x**	Unknown error

* -*x* represents any other number.

Troubleshooting Recipe Script Functions

To retrieve the error code of a Recipe Function, it must be equated to an InTouch integer tagname.

Example:

```
ErrorCode = RecipeLoad(FileName, UnitName, RecipeName);
```

The **RecipeLoad()** function will set the value of the tagname **ErrorCode** to **0** if it is successful. If the **RecipeLoad()** fails, it will set the analog tagname, **ErrorCode**, to the number for the specific error condition. The following is a listing of the possible Error Codes and their corresponding error messages and descriptions:

Value	Error Message	Description
0	**Success**	The called recipe function executed successfully.
-1	**No Such Recipe Template**	The specified recipe template filename does not exist.
-2	**View Not Active**	The recipe function called by another program cannot execute because WindowViewer is not running.
-3	**Out of Memory**	There is not enough memory to complete the current activity.
-4	**Line too long in recipe template file**	A line in the recipe template file has exceeded the maximum allowed length.
-5	**Truncated line in the recipe file**	A line in the recipe template file has been truncated.
-6	**Not a valid recipe template file**	The specified filename is not a valid .CSV recipe template file. 📖 For more information on unit or recipe names, see your online *Recipe Manager User's Guide*.
-7	**Expecting "unit" or "recipe"**	A unit name or recipe name is missing from the recipe template file. 📖 For more information on unit or recipe names, see your online *Recipe Manager User's Guide*.
-8	**No units defined in recipe template file**	No units have been defined in the recipe template file Units Definition template.
-9	**Recipe name not found in recipe template file**	The specified recipe name is not defined in the recipe template file.
-10	**Unit name not found in recipe template file**	The specified unit name is not defined in the unit definition template file

Value	Error Message	Description
-12	**Expecting "Analog", "Discrete", "Message"**	An incorrect type has been entered for an item in the recipe template file. Valid types are Analog, Discrete or Message only.
-13	**Type of tagname mismatches "Analog","Discrete", "Message"**	The tagname specified is incorrect for the item type, e.g., a recipe item is defined as Analog and a message tagname has been defined in the unit for it.
-14	**Invalid discrete value, expecting "0", "1"**	An incorrect value has been entered for a Discrete in the recipe template file. The only valid values for Discretes are 0 or 1.
-15	**Unable to open temporary file**	The temporary file cannot be opened; could possibly be due to inadequate disk space.
-16	**Write error while saving recipe template file**	An error has occurred while saving the recipe template file.
-17	**User did not select**	The user selected **Cancel** in the **Select a Recipe** dialog box instead of a recipe name.
-19	**Recipe template in use by another application**	The recipe template file specified is open and, therefore, cannot be accessed by WindowViewer.

Displaying Error Code Messages

Each Recipe Function returns a number that represents the error condition for the function. By using the **RecipeGetMessage()** function in an InTouch Data Change script, the corresponding error code can be written to an analog tagname and the associated error code message can be written to a message tagname.

To do so, the following Data Change script would be used:

```
RecipeGetMessage(ErrorCode, ErrorMessage, 131);
```

This script will automatically execute whenever the value of the analog tagname **ErrorCode** changes. When this script executes, the **RecipeGetMessage()** function will read the current numeric value of the tagname **ErrorCode** and return the message associated with that value to the tagname **ErrorMessage**.

SPC DDE Item Names

DDE items are available for obtaining Dataset information and for controlling chart operations. The *application name* is SPC. The *topic name* is the Dataset name.

Note All SPC DDE Items are used in the same way for distributed SPC as they are for non-distributed SPC, except the *topic name* must be an InDirect Dataset name for distributed SPC.

SPC Control and Display DDE Items

The Control and Display DDE Items are used to control and display information about the topic Dataset. Control DDE items are shared by all nodes. They are the dataset values for the collected product of the remote dataset. Display DDE items are local to each node. They are the sample values for the displayed product on the local node.

Sample modifications can be applied to collected and displayed products of locally any datasets. Modification made by clicking on a chart display will affect the displayed product. The SPC DDE items modify the collected product. The displayed product and the collected product have their own current samples which is the most recently recorded sample.

Alarms are evaluated and stored for collected and displayed products. They are only reported during runtime for the collected product.

Note With the addition of displayed and collected products, many SPC DDE Items apply only to the collected product. These items are flagged in the following lists by an asterisk (*) preceding the SPC DDE Item name.

Item Name	DDE Type	Access	Description
AutoCollection	Discrete	R/W	Enables/disables automatic data collection.
***CalculateControlLimits**	Discrete	R/W	Set to 1 to start control limit calculation.
DatasetName	Message (32)	R/W	Sets the Dataset Name used by an Indirect Dataset.
HistogramLCL	Real	RO	Displays the Histogram's Lower Control Limits based on population.
HistogramUCL	Real	RO	Displays the Histogram's Upper Control Limits based on population.
Kurtosis	Real	RO	Distribution shape of Histograms.
LastSampleDisplayed	Integer	R/W	Sets the last sample number displayed by the Dataset.
***ManualInputDialog**	Discrete	R/W	Set to 1 to display built-in Manual Input Dialog Box.
MeasurementsPerSample	Integer	RO	Displays the configured number of measurements per sample.
NewProduct	Message (32)	R/W	Used to create new Product Name.

Item Name	DDE Type	Access	Description
***ProductCollected**	Message (32)	R/W	Changes the Product Name collected by the Dataset.
ProductDisplayed	Message (32)	R/W	Changes the Product Name displayed by the Dataset.
SampleSize	Integer	RO	Sample size for NP Dataset.
SamplesPerControlChart	Integer	R/W	Sets the number of samples displayed in a Control Chart.
SamplesPerHistogram	Integer	R/W	Sets the number of samples displayed in a Histogram.
SamplesPerLimitCalc	Integer	R/W	Sets the number of samples used in a control limit calculation.
SamplesPerPareto	Integer	R/W	Sets the number of samples used in a Pareto Chart display.
SelSPCOutSpecMsg	Message	RO	Alarm Message tag for "Sample outside specification Limit."
Skewness	Real	RO	Displays the variance from the mean on Histograms.
SPCAllowSampDelMod	Discrete	R/W	Toggles right click menu options Delete and Modify sample options on and off.
StartCollection	Discrete	R/W	Set to 1 to start an Auto collection cycle.

SPC Current Sample DDE Items

All Current Sample DDE Items pertain to the last collected sample of a given Dataset. They can be used to change the raw data and the limits associated with the Dataset Name. To change information about the current sample, you must write to the appropriate DDE Item then set the **CurrentUpdate** DDE Item to 1. This will have the effect of re-entering the sample and will cause any required calculations to be performed. The SPC program will reset the **CurrentUpdate** DDE Item to **0** after the sample has been entered. Once the next sample to be collected has started a collection cycle, the current sample DDE items can no longer be updated.

Current sample DDE items are shared among all nodes. These item values represent the last collected sample of a collected product.

For distributed SPC, initially all the values are set to zero. SPC connects to the database and checks for new data every 5 seconds. The item values are updated whenever new information is found. Modifications to the current sample values are buffered locally until the **CurrentUpdate** item is set to 1. Then the values are placed in a current sample packet and sent to the remote dataset node for analysis and storage. Current sample modifications that indicate a different collected product and a current sample number that is not the last recorded sample will be rejected by the Server.

Note With the addition of displayed and collected products, all of the "Current" SPC DDE Items apply only to the collected product.

Item Name	DDE Type	Access	Description
CurrentCauseCode	Integer	R/W	Sets the Special Cause Code number for the current sample.
CurrentCauseString	Message (128)	RO	Displays the description of the Special Cause Code number for the current sample.
CurrentComment	Message (50)	R/W	Used to read/write any miscellaneous comments associated with the current sample.
CurrentCp	Real	RO	Displays the capability for the current sample.
CurrentCpk	Real	RO	Displays the centered capability for the current sample.
CurrentDate	Message (8)	R/W	Sets the Date for the current sample in the format **DD/MM/YY**. If incorrectly entered, defaults to the current Date.
CurrentFlag	Discrete	R/W	Sets a Flag for the current sample.
CurrentIgnoreValue	Discrete	R/W	Sets the current sample to be ignored when the Control Chart is AutoScaled.
CurrentM*x*	Real	R/W	Sets the individual measurement value for the current sample. (*x*=1 to 25.)
CurrentR	Real	RO	Displays the range for the current sample.
CurrentRBar	Real	R/W	Sets the average range at the current sample.
CurrentRLCL	Real	R/W	Sets the range Lower Control Limit.

Item Name	DDE Type	Access	Description
CurrentRUCL	Real	R/W	Sets the range Upper Control Limit.
CurrentSample	Real	RO	Displays the value of the last sample point (i.e., X, C, P).
CurrentSampleBar	Real	R/W	Sets the current sample average at this sample point.
CurrentSampleNumber	Integer	RO	Displays the last sample number collected.
CurrentTarget	Real	R/W	Sets the target value at this sample point.
CurrentTime	Message (8)	R/W	Sets the Time for the current sample in the format **HH:MM:SS**. If incorrectly entered, defaults to the current Time.
CurrentUpdate	Discrete	R/W	To change information entered about the sample in any of the current fields, set this Item to 1.
CurrentXLCL	Real	R/W	Sets current sample Lower Control Limit (LCL).
CurrentXLSL	Real	R/W	Sets current sample Lower Specification Limit (LSL).
CurrentXUCL	Real	R/W	Sets current sample Upper Control Limit (UCL).
CurrentXUSL	Real	R/W	Sets current sample Upper Specification Limit (USL).
SPC2L3Out2SD	Integer	RO	Alarm counter for Alarm "2 of the last 3 samples outside of 2 standard Deviation SS."
SPC2L3Out2SDMsg	Message	RO	Alarm Message tag for "2 of the last 3 samples outside of 2 standard Deviation SS."
SPC4L5Out1**SD**	Integer	RO	Alarm counter for Alarm "4 of the last 5 samples outside of 1 standard Deviation SS."
SPC4L5Out1**SDMsg**	Message	RO	Alarm Message tag for "4 of the last 5 samples outside of 1 standard Deviation SS."
SPCConSampAltUpDn	Integer	RO	Alarm counter for Alarm "Consecutive samples Alternating Up and Down."
SPCConSampAltUpDnMsg	Message	RO	Alarm Message tag for "Consecutive samples Alternating Up and Down."
SPCConSampln1SD	Integer	RO	Alarm counter for Alarm "Consecutive samples Inside 1 standard Deviation."
SPCConSampln1SDMsg	Message	RO	Alarm Message tag for "Consecutive samples Inside 1 standard Deviation."

Item Name	DDE Type	Access	Description
SPCConSamplncDec	Integer	RO	Alarm counter for Alarm "Consecutive samples Increasing or Decreasing."
SPCConSamplncDecMsg	Message	RO	Alarm Message tag for "Consecutive samples Increasing or Decreasing."
SPCConSampOneSideCL	Integer	RO	Alarm counter for Alarm "Consecutive samples on one side of centerline."
SPCConSampOneSideCLMsg	Message	RO	Alarm Message tag for "Consecutive samples on one side of centerline."
SPCConSampOut1**SD**	Integer	RO	Alarm counter for Alarm "Consecutive samples outside 1 standard Deviation."
SPCConSampOut1SDMsg	Message	RO	Alarm Message tag for "Consecutive samples outside 1 standard Deviation."
SPCNLNOutNSD	Integer	RO	Alarm counter for Alarm "? Of the last ? samples outside ? standard deviations."
SPCNLNOutNSDMsg	Message	RO	Alarm Message Tag for "? Of the last ? samples outside ? standard deviations."
SPCNLNOutNSDSS	Integer	RO	Alarm counter for Alarm "? Of the last ? samples outside ? standard deviations SS."
SPCNLNOutNSDSSMsg	Message	RO	Alarm Message tag for "? Of the last ? samples outside ? standard deviations SS."
SPCOutRCtrl	Integer	RO	Alarm counter for the Range Chart Alarm "Range outside Control Limit."
SPCOutRCtrlMsg	Message	RO	Alarm Message for the Range Chart Alarm "Range outside Control Limit."
SPCOutXCtrl	Integer	RO	Alarm counter for the X Chart Alarm "Sample outside Control Limit."
SPCOutXCtrlMsg	Message	RO	Alarm Message for the X Chart Alarm "Sample outside Control Limit."
SPCOutSpec	Integer	RO	Alarm counter for Alarm "Sample outside specification Limit."
SPCOutSpecMsg	Message	RO	Alarm Message tag for "Sample outside specification Limit."
SPCResetAlarmCounters	Discrete	R/W	Resets all alarm counters.

SPC Manual Input DDE Items

The Manual Input DDE Items are used to create custom manual input windows. To use the manual input items, set the values of the appropriate items and then set the **MI_Save** DDE Item to 1. This will cause the information in the other MI fields to be entered as a new sample. The SPC program will reset the **MI_Save** DDE Item to 0 (zero) after the sample has been entered.

For distributed SPC, manual input DDE items are private to each node. The values are buffered locally at each node until the DDE item **MI_Save** is set to 1. Once **MI_Save** is set to 1, the values are placed in a manual input packet and sent to the remote dataset node for analysis and storage.

Note With the addition of displayed and collected products, all of the "Manual" SPC DDE Items apply only to the collected product.

Item Name	DDE Type	Access	Description
MI_CauseCode	Integer	R/W	Sets the Special Cause Code number for the manually input sample.
MI_CauseString	Message (127)	RO	Displays the description of the Special Cause Code number input for the sample.
MI_Comment	Message (50)	R/W	Used to read/write any miscellaneous comments entered for the sample.
MI_Date	Message (8)	R/W	Sets the Date for the current sample. The Date must be entered in the format **DD/MM/YY**. If incorrectly entered, the Date will default to the current Date.
MI_Flag	Discrete	R/W	Sets a flag for the manually input sample.
MI_IgnoreValue	Discrete	R/W	Sets the current sample to be ignored when the Control Chart is AutoScaled.
MI_M*x*	Real	R/W	Sets the value for the designated manually input measurement (*x*=1 to 25).

Item Name	DDE Type	Access	Description
MI_Save	Discrete	R/W	Saves the information manually entered in the other MI fields as a new sample. **Note** When the **MI_Save** item is set to 1, the value of all **MI** items are written to the respective **Current** DDE Items and the **CurrentSampleNumber** item is indexed by 1.
MI_Time	Message (8)	R/W	Sets the Time for the current sample. The Time must be entered in the format **HH:MM:SS**. If incorrectly entered, the Time will default to the current time.

SPC Selection DDE Items

The Selection DDE Items can be used to view detailed information about any sample. The DDE Item Selection is used to enter the number of the sample to be displayed. Once entered, the SPC program will update all of the other Selection Items with the detailed information for the Selection sample number.

Old data cannot be changed, but Special Cause Codes, Flags and/or Comments can be added by setting the appropriate items then setting the **SelectionUpdate** item to 1.
This will cause the selection sample record to be modified with the new values. The SPC program will reset the **SelectionUpdate** DDE item to 0 (zero) after the updated sample has been entered.

For distributed SPC, selected sample DDE items are private to each node. They are the sample values recorded by the remote node for a specified sample number of the collected product. When the Selection DDE item is set to a sample number, the sample information is retrieved from the remote node's sample file. Old data cannot be changed, but Special Cause Codes, Flags, and Comments can be added by changing the appropriate DDE item and setting the **SelectionUpdate** item to 1. When **SelectionUpdate** is set to 1, the Special Cause Code, Comment, Flag, and Ignore Value items are sent to the remote node in a packet for storage.

Note With the addition of displayed and collected products, all of the "Selection" SPC DDE Items apply only to the collected product.

Item Name	DDE Type	Access	Description
Selection	Integer	R/W	Setting this item to a sample number will update all of the selection items with the appropriate data for that sample.
SelectionCauseCode	Integer	R/W	Sets the Special Cause Code number for the selected sample.
SelectionCauseString	Message (128)	RO	Displays the description of the entered Special Cause Code.
SelectionComment	Message (50)	R/W	Used to read/write any miscellaneous comments entered for the selected sample.
SelectionCp	Real	RO	Displays the capability for the selected sample.
SelectionCpk	Real	RO	Displays the centered capability for the selected sample.
SelectionDate	Message (8)	RO	Displays the date for the selected sample.
SelectionFlag	Discrete	R/W	Sets a Flag of the selected sample.
SelectionIgnoreValue	Discrete	R/W	Sets the selected sample to be ignored when the Control Chart is AutoScaled.
SelectionMx	Real	RO	Displays the value for the individual measurements (*x*=1-25) comprising the sample.
SelectionProduct	Message (32)	RO	Displays the Product Name for the selected sample.

Item Name	DDE Type	Access	Description
SelectionRUCL	Real	RO	Displays the range **UCL** for the selected sample.
SelectionRLCL	Real	RO	Displays the range **LCL** for the selected sample.
SelectionR	Real	RO	Displays the **range** for the selected sample.
SelectionRBAR	Real	RO	Displays the **average** range at the selected sample.
SelectionSample	Real	RO	Displays the value of the selected sample point.
SelectionSampleBar	Real	RO	Displays the selected sample average at the selected sample point.
SelectionTarget	Real	RO	Displays the **target** value at the selected sample.
SelectionTime	Message (8)	RO	Displays the **Time** for the selected sample.
SelectionUpdate	Discrete	R/W	Updates changes in Selection fields.
SelectionXUSL	Real	RO	Displays sample Upper Specification Limit.
SelectionXLSL	Real	RO	Displays sample Lower Specification Limit.
SelectionXUCL	Real	RO	Displays sample Upper Control Limit.
SelectionXLCL	Real	RO	Displays sample Lower Control Limit.
SelSPC2L3Out2SDMsg	Message	RO	Alarm Message tag for "2 of the last 3 samples outside of 2 standard Deviations SS."
SelSPC4L5Out1SDMsg	Message	RO	Alarm Message tag for "4 of the last 5 samples outside of 1 standard Deviation SS."
SelSPCConSampAltUpDnMsg	Integer	RO	Alarm message for Alarm "Consecutive samples Alternating Up and Down."
SelSPCConSampln1SDMsg	Message	RO	Alarm Message tag for "Consecutive samples Inside 1 standard Deviation."
SelSPCConSamplncDecMsg	Message	RO	Alarm Message tag for "Consecutive samples Increasing or Decreasing."
SelSPCConSampOneSideCLMsg	Message	RO	Alarm Message tag for "Consecutive samples on one side of centerline."
SelSPCConSampOut1SDMsg	Message	RO	Alarm Message tag for "Consecutive samples outside 1 standard Deviation."
SelSPCNLNOutNSDMsg	Message	RO	Alarm Message Tag for "? Of the last ? samples outside ? standard deviations."

Item Name	DDE Type	Access	Description
SelSPCNLNOutNSDSSMsg	Message	RO	Alarm Message tag for "? Of the last ? samples outside ? standard deviations SS."
SelSPCOutRCtrlMsg	Message	RO	Alarm Message for the Range Chart Alarm "Range outside Control Limit."
SelSPCOutXCtrlMsg	Message	RO	Alarm Message for the X Chart Alarm "Sample outside Control Limit."
SelSPCOutSpecMsg	Message	RO	Alarm Message tag for "Sample outside specification Limit."

Note Multiple Indirect Datasets can be setup and linked to the same real Dataset. Then the Selection value of each Indirect Dataset can be set to a different sample number. This allows you to view detail information of multiple samples within a Dataset.

Troubleshooting SQL Script Functions

All SQL Functions return a *ResultCode* that can be used for troubleshooting. The **SQLErrorMsg()** function returns the Error Message associated with the *ResultCode*.

Example:

```
ErrorMsg=SQLErrorMsg(ResultCode);
```

where: **ErrorMsg** is a memory message tag.
ResultCode is an integer value obtained from a previous SQL function.

Result Code Error Messages

For Result Codes that are not documented here, please refer to your specific database documentation and be sure to check the Wonderware Logger for any additional information.

The **SQLErrorMsg()** function will set the value of the InTouch message tagname *ErrorMsg*. The following is a listing of some of the possible SQL Result Codes and their corresponding error messages and descriptions:

Result Code	Error Message	Description
0	**No errors occurred**	The command was successful
-5	**No more rows to fetch**	The last record in the table has been reached
-1001	**Out of memory**	There is insufficient memory to perform this function
-1002	**Invalid connection**	The ConnectionId passed to the function is not valid
-1003	**No bind list found**	The specified Bind List name does not exist
-1004	**No template found**	The specified Table Template name does not exist.
-1005	**Internal Error**	An internal error occurred. Call Technical Support.
-1006	**String is null**	Warning - the string read from the database is null.
-1007	**String is truncated**	Warning - the string read from the database is longer than 131 characters and is truncated on a select.
-1008	**No Where clause**	There is no Where clause on Delete.

Result Code	Error Message	Description
-1009	**Connection failed**	Check Wonderware Logger for a more detailed description of the failed connect.
-1010	**The database specified on the DB= portion of the connect string does not exist**	The specified database does not exist.
-1011	**No rows were selected**	A **SQLNumRows()**, **SQLFirst()**, **SQLNext()**, or **SQLPrev()** command was attempted without executing a **SQLSelect()** command first.
-4149	**The connection, statement, or query handle you provided is not valid**	Column type may be incorrectly defined. For example, if a Table Template is defined with a column type of character instead of char for a dBASE file, an error will be returned.

Specific Database Error Messages

Oracle

Error Message	Solution
ORA-03112 - Host String Syntax error	If you are not running **NETINIT.EXE**, place it in the Windows Startup group. If you are running **NETINIT.EXE** and want to establish more than one connection or session, you will need to allocate memory. Do this by setting the **WIN_REMOTE_SESSIONS** parameter in the **CONFIG.ORA** file equal to the number of required connections. The following example will allocate memory for 4 connections: **WIN_REMOTE_SESSIONS=4**
ORA-3121 - No interface driver connected	Start the SQL*NET TSR appropriate for your networking system **before** entering Windows and using InTouch SQL Access.
ORA-6435 - NetBIOS: Unable to add local name to name table	The network software (Novell, LAN Manager, etc.) must be running.
ORA-09301 - Local kernel only supported in standard mode	Specify the server name (**"SRVR="**) in the connection string.
ORA-06430 - Unable to make connection	Verify that the attributes in the connection string are accurate and in correct order.

Sybase or Microsoft SQL Server

Error Message	Solution
You cannot have more than one statement active at a time	You are trying to execute a SQL command after executing a **SQLSelect()**. Execute a **SQLEnd()** to free system resources from the **SQLSelect()** or; Use a separate ConnectionId for the second statement.
There is not enough memory available to process the command	Try rebooting the client workstation.
Invalid object name table name	The table name does not exist in the database you are using. Try DB=database name.

dBASE

Error Message	Solution
File or DLL not found	For Windows, the QEDBF.DLL must either be in your current directory or in the windows system directory in your DOS path.
Invalid connection	Make sure you have appropriate DLLs in your path. For DBF support, you will need QLDBF.DLL.
The connection or statement handle you provided is not valid	For more information, check the Wonderware Logger. There may be a syntax error in your SQL statement.

Index

$

.

A

C

D

E

F

G

H

I

T

W