AVEVA Application Server
formerly Wonderware

Application Server Scripting Guide

Publication date: Friday, February 21, 2020

Contact Information

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

https://sw.aveva.com/

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

Contents

CHAPTER 1

Common Scripting Environment

This section describes common styles, syntax, commands, and behaviors of scripts within AVEVA™ Application Server, formerly Wonderware.

Script Editing Styles and Syntax

Application Server supports two types of scripts:

- Simple scripts can perform assignments, comparisons, simple math functions, and similar actions. Simple scripts are described in this section.
- Complex scripts can perform logical operations using conditional branching with IF-THEN-ELSE type control structures. For more information about complex control structures, see *QuickScript .NET Control Structures* on page 107.

Both single and multi-line comments are supported. Single-line comments start with a " ' " in the line but require no ending " ' " in the line. Multi-line comments start with a "{" and end with a "}" and can span multiple lines.

White space rules apply for space and indention. Indent using spaces, or the `TAB` key. Individual statements are indicated by a semicolon marking the end of the statement.

Required Syntax for Expressions and Scripts

The syntax in scripts is similar to the algebraic syntax of a calculator. Most statements are presented using the following form:

```
a = (b - c) / (2 + x) * xyz;
```

This statement places the value of the expression to the right of the equal sign (=) in the variable location named "a."

- A single entity must appear to the left of the assignment

  operator =.
- The operands in an expression can be constants or variables.
- Statements must end with a semicolon (;).

Entities can be concatenated by using the plus (+) operator. For example, if a data change script such as the one below is created, each time the value of "Number" changes, the indirect entity "Setpoint" changes accordingly:

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

Where the result is "`Setpoint1`."

Simple Scripts

Simple scripts implement logic such as assignments, math, and functions. An example of this type of scripting is:

```
React_temp = 150;
ResultTag = (Sample1 + Sample2)/2;
{this is a comment}
```

Script Execution Types

This section describes the script execution types supported by InTouch OMI.

Startup Scripts

Startup scripts are called when an object containing the script is loaded into memory, such as during deployment, platform, or engine start.

Startup instantiates COM objects and .NET objects. Depending on load and other factors, assignments to object attributes from the Startup method may fail. Attributes that reside off-object are not available to the Startup method.

OnScan Scripts

OnScan scripts are called the first time an AppEngine calls this object to execute after the object's scan state changes to OnScan. The OnScan method initiates local object attribute values and provides more flexibility in the creation of .NET or COM objects.

Attributes that are off-engine are not available to the OnScan method.

Execute Scripts

Execute scripts are called each time the AppEngine performs a scan and the object is OnScan.

The Execute script method is the workhorse of the scripting execution types. Use the Execute method for your run-time scripting to ensure that all attributes and values are available to the script.

If the **Quality** check-box is checked, the Execute method is similar to InTouch® scripts with the following conditional trigger types:

- Periodic: When going OnScan, a script with a periodic trigger executes immediately (at the next scheduled scan period of the AppEngine). It then executes periodically whenever the elapsed time evaluates as true.
- Data Change: Executes when a data value or quality changes between scans.

For the following trigger types, data changes between each scan are not evaluated, only the value at the beginning of each script is used for evaluation purposes. For example, if a Boolean attribute changes from True to False to True again during a scan cycle, this change is not evaluated as a data change as the value is True at the beginning of each scan cycle.

- OnTrue: Executes if the expression validates from a false on one scan to a true on the next scan.
- OnFalse: Executes if the expression validates from a true on one scan to a false on the next scan.

These scripts also have time-based considerations. A trigger period of 0 means that the script executes every scan.

Time-based scripts, WhileTrue, WhileFalse, and Periodic are evaluated and executed based on the elapsed time from a timestamp generated from the previous execution, not on an elapsed time counter. It is possible that a change in the system clock can change the interval between execution of these scripts.

- WhileTrue: Executes scan to scan as long as the expression validates as true at the beginning of the scan.
- WhileFalse: Executes scan to scan as long as the expression validates as false at the beginning of the scan.

For example, a periodic script is set to run every 60 minutes. The script executes at 11:13 AM. We expect it to execute 60 minutes later at 12:13 PM. However, a time synchronization event occurred and the node's time is adjusted from 11:33 AM to 11:30 AM.

The script still executes when the system time reaches 12:13 PM. But because of the time change, the actual (True) time period that elapsed between executions is 63 minutes.

OffScan Scripts

OffScan scripts are called when the object is taken OffScan. This script type is primarily used to clean up the object and account for any needs to address as a result of the object no longer executing.

If an object is taken OffScan, either directly, or indirectly because its engine is taken OffScan, all in-progress asynchronous scripts for that object are requested to shut down by setting a Boolean shutdown attribute for the script to true. A well-written script checks this attribute before and after time-consuming operations. If the script takes more than 30 seconds to complete, a warning appears in the logger that the script is not responding to the shutdown command. However, the script is allowed to complete and is not terminated by force. This all takes place on the engine's main thread and could potentially hang the engine. During this time, the script might also time out and as a result exit before executing all its logic.

Shutdown Scripts

Shutdown scripts are called when the object is about to be removed from memory, usually as a result of the AppEngine stopping. Shutdown scripts are primarily used to destroy COM objects and .NET objects and to free memory.

Deployment Scripts

Deploying objects is both a critical and a load-intensive process for a Galaxy. Implementing scripting in the Startup and OnScan methods can adversely affect a Galaxy's deployment and redundancy performance.

While objects are being deployed, their Startup and, if deployed OnScan scripts are executed. These scripts must complete within the deployment time-out period for the deployment to be successful.

Placing large numbers of scripts, or scripts that require heavy processing power into the Startup or OnScan script methods can slow or cause a deployment or failover to fail. In addition to the load that is placed on the system at deployment time, the type of scripting done in the Startup and OnScan methods is also important becaus e these scripts execute in a sequence.

During deployment and restart, the Startup and OnScan script methods do not execute objects based on execution order. Objects are started up and placed on scan based on their alphanumeric tag name within their hosting Area.

Follow the recommendation below for each type of script method to help determine what scripting practices to follow in each script method.

Do not place the following types of scripting in the Startup or OnScan methods:

- Database access
- File system access to .csv, .xml, .txt, and other file types
- Off-object referencing
- Dynamic referencing

Working with QuickScript Editor Features

The QuickScript editor provides a number of features to enhance scripting speed and accuracy.

Color Indicators for Script Elements

The QuickScript .NET editor uses different text colors to identify different script elements. The following table shows the text colors associated with script elements.

Element	Color
Keywords	Blue Syntax highlighted while typing.
Comments (both single line and multi-line)	Green Syntax highlighted while typing.
Strings	Purple Syntax highlighted while typing.
Function names, numeric constants, operators, semicolons, dim variables, alias variables, and so on	Black See descriptions for Attribute names and Reserved words.
Attributes, InTouch Tags, Reference Strings	Maroon, bold face
Reserved words	Red, non-bold face
.NET type names	Teal, non-bold face

Autocomplete

QuickScript autocomplete incorporates several features for use while authoring object and client scripts:

- Provides an autocomplete Attribute reference when you type a generic object name, such as "me." Run-time attributes appear in an autocomplete list box. Typing "InTouch:" displays an autocomplete list of tagnames from the most recently selected ViewApp template.
- Provides method parameter help in an autocomplete list box including context-specific suggestions covering definitions, keywords, script elements, and programmatic constructs such as try ... catch or while ... endwhile.
- Automatic word completion of Attribute references, methods, programmatic constructs, and other script elements.

These features serve as convenient documentation of method parameters and scripting syntax as well as an enhanced input method.

Autocomplete displays a context-sensitive list of options for script elements, keywords, object and attribute names, and programmatic constructs. Press Ctrl+space to display all available autocomplete options and variables for the selected location in the script. You can identify the context from the icons displayed with the list items.

Icon	Represents
	MxBoolean attribute
	MxInteger attribute
	MxFloat attribute
	MxDouble attribute
	MxString attribute
	MxTime attribute
	MxElapsedTime attribute
	MxReference attribute
	MxStatus attribute
	MxDataTypeEnum attribute
	MxSecurityClassification attribute
	MxDataQuality attribute
	MxQualifiedEnum attribute
	MxQualifiedStruct attribute
	MxInternationalizedString attribute
	.Net Method
	.Net Property

Icon	Represents
	.Net Field or Variable
	.Net Namespace
	.Net Struct
	.Net Class
	.Net Interface
	.Net Enumeration
	.Net Enum Value
	QuickScript Keyword
	Contained object name, or any partial attribute name such as a attribute, field attribute, or primitive that has a dot in the name, or any attribute of Mx type MxNone, or if there are several type choices among objects and attributes. If the attribute cannot be exactly or unambiguously returned, this icon will appear. Partial name example: For me.alarm.a1, typing "me.alar" will show the blue ball icon for alarm. MxNone example: input/output extension attribute WriteValue.
	Rectangle
	Rounded rectangle
	Line
	Horizontal or vertical line
	Text
	Ellipse

Icon	Represents
	Curve
	Closed curve
	Button
	Polygon
	Polyline
	Connect
	Image
	Group or embedded symbol
	Alarm control
	Edit box
	Arc
	Pie
	Chord
	Circle
	Status
	Radio buttons
	Checkbox

Icon	Represents
	Edit box
	Combo box
	Calendar
	Date picker
	List box

Accepting Autocomplete Suggestions

Insert an item at the editor caret from the autocomplete list box—without an end line or tab appended—by doing one of the following:

- Double-click the item.
- Highlight (select) the item and press the **Enter** key or the **Tab** key.

Type a space, period, comma, open or closed parenthesis, or other punctuation used in the QuickScript .NET programming language (: ; [ ] = < > - + / *), and the item highlighted in the autocomplete list box will be inserted at the editor caret with the additional character appended.

Multi-level Undo and Redo

You can selectively undo a history of changes to your script. The number of changes that can be undone is limited only by the amount of available memory.

An undone change can be redone. Redo mirrors undo changes.

A single undo typically is comprised of sequences of typing or deleting, which can be interrupted by interaction with an autocomplete list or by moving the cursor with the mouse, or by clicking elsewhere in the script.

All pending undo and redo actions will be lost if you close the object editor, switch to another script within the object editor, or switch among Startup, OnScan, Execute, OffScan, and Shutdown scripts.

Dynamic Referencing Considerations

Dynamic reference scripting is one the biggest causes of deployment failures of StartUp and OnScan execution types.

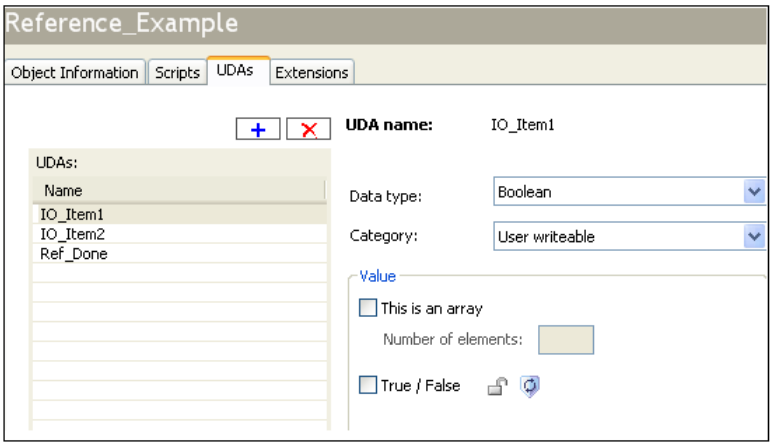
Rather than placing dynamic referencing scripts in the Startup or OnScan methods, perform dynamic referencing in the Execute method. There are several advantages to using the Execute method with dynamic reference scripting:

- Deployment is faster.
- Deployment is more reliable.
- Deterministic execution order is guaranteed.

- Off-object and off-engine attributes are available.
- After a failover occurs, the startup of the redundant engine is more stable and can be faster.

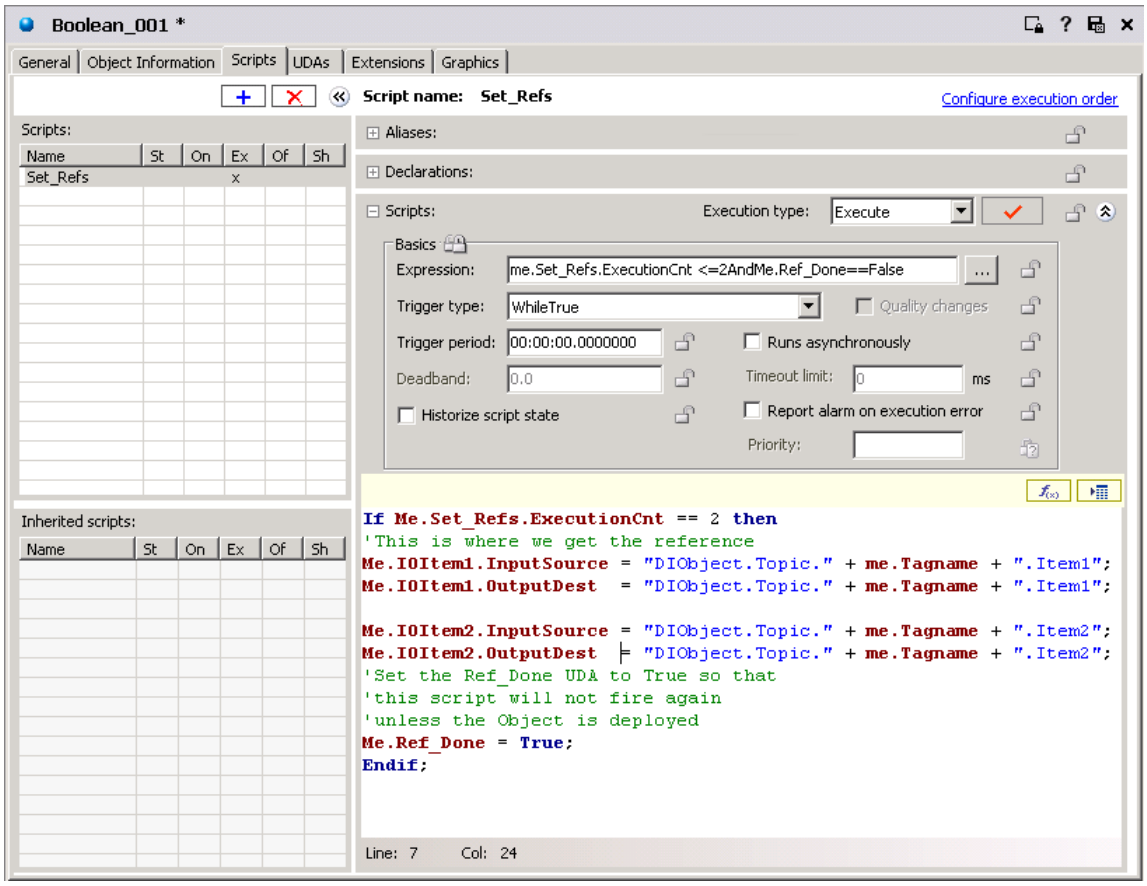
To create a simple dynamic reference script example

1. Create a Boolean attribute.



   The attribute shows if the referencing script is complete. In this example you create `Ref_Done`. `IO_Item1` and `IO_Item2` are the I/O points referenced in this example.

2. Create the script. The script in this example is called `Set_Refs`. The script has a trigger type of `WhileTrue` with a `0` trigger period.



The script is shown below:

```
If Me.Set_Refs.ExecutionCnt == 2 then
Me.IOItem1.InputSource = "DIObject.Topic." + me.Tagname + ".Item1";
Me.IOItem1.OutputDest = "DIObject.Topic." + me.Tagname + ".Item1";
Me.IOItem2.InputSource = "DIObject.Topic." + me.Tagname + ".Item2";
Me.IOItem2.OutputDest = "DIObject.Topic." + me.Tagname + ".Item2";
Me.Ref_Done = True;
Endif;
```

This script allows the system to stabilize after going on scan before setting the references. The script executes on the first two scans of the object when the Boolean attribute `Ref_Done` is false.

As the script is executed, a check is made against the execution count. If the count equals 2, the script performs the referencing operations. After the reference attributes are set on the attributes, the `Ref_Done` attribute is set to True. At this point the expression for the script is no longer true.

The three attributes set in this script are checkpointed, eliminating the need to run this script except on deployment. The next time the object is started, placed on scan, or failed over, there is no need to recreate the references to the items.

Run-Time Client Script Behavior

In Advanced Communication Management, script references to InTouch tags and object attributes are suspended from receiving data changes when the application window containing embedded ArchestrA® objects is minimized in InTouch WindowViewer. Suspending data updates to hidden objects reduces the amount of network traffic and improves the overall performance of a client application.

While Showing scripts of embedded symbols do not execute during the period when the window containing the symbols is minimized. Script execution resumes after restoring or maximizing a window that had been previously closed or minimized.

Opening a Client Application Window

In Advanced Communication Management, when a client application window containing embedded ArchestrA objects opens in WindowViewer, the following script events occur:

- Register all ArchestrA and InTouch references used in embedded symbol scripts, if not registered already.
- Advise all ArchestrA and InTouch references in embedded symbol scripts within the window, if not advised already.
- Execute the OnShow script on all embedded symbol scripts within the window.
- Execute named scripts if their trigger conditions are met.

Closing a Client Application Window

In Advanced Communication Management, when a client application window containing embedded ArchestrA objects is closed, the following script events occur:

- Execute OnHide scripts of all embedded symbols within the window.
- Stop running client scripts.
- Unadvise all ArchestrA and InTouch references in the Window if there are no other open windows using the references.
- Unregister all ArchestrA and InTouch references in the Window if there are no other open windows using the references.

Minimizing a Client Application Window

In Advanced Communication Management, when an open window containing embedded ArchestrA objects is minimized in WindowViewer, the following script events occur:

- Stop running client scripts associated with ArchestrA objects embedded in the window.
- Unadvise all ArchestrA and InTouch references in the Window if there are no other open windows using the references.
- OnHide scripts of embedded symbols do not execute when a window is minimized.

Maximizing or Restoring a Client Application Window

In Advanced Communication Management, after maximizing or restoring a window from WindowViewer that had been previously minimized or closed, the following script events occur:

- Advise all ArchestrA and InTouch script references in the window, if not advised already.
- Execute named scripts if their trigger conditions are met.

Visual Indication of Script Errors

Verification errors in script text are marked with a red "squiggly" underline. The underline appears after approximately 2.5 seconds of keyboard inactivity.

Hovering over the error with the mouse cursor will display the error message as a tooltip. The tooltip error message is identical to the message shown when clicking the script verification button.

Note: In addition to error tooltips, the script editor will also display the variable name and type in a tooltip when hovering over a variable name in the script.

In some cases, more than one error will be underlined. This is not always possible because some errors prevent the compiler from continuing past the error.

Line Numbers

The script editor displays line numbers in the left margin.

- Line numbers of up to four digits will display when the script editor is not zoomed.
- The line number may appear clipped for scripts longer than 9999 lines or when the script editor is zoomed.
- Use the right-click context menu **Go To** function to go to a specific line in the script.

Log Functions

QuickScript .NET functions include several log functions to capture and display information in the logger under different log flags.

- *LogCustom()* on page 69
- *LogError()* on page 70
- *LogMessage()* on page 71
- *LogTrace()* on page 71
- *LogWarning()* on page 72

Important: To use the LogCustom function, you must enable Log Custom in the System Management Console (SMC) Log Flag Editor. To use the LogTrace function, you must enable Log Trace in the SMC Log Flag Editor.

CHAPTER 2

QuickScript .NET Functions

For information about other functions in this category, see third-party documentation.

Keep in mind the following limitations when you use the script functions:

- Be aware of the .NET datatypes.
- Starting a GUI application from within a server script is not supported.
- Although QuickScript supports import libraries built with .NET CLR version 2.0.50727, it does not support any of the new language features introduced with .NET 2.0, such as generics.

Script Functions

This section describes the script functions included in the InTouch OMI development environment. The function documentation is organized into a set of folders that represents the same organization of the functions in the Script Function Browser.

- *Graphic Client Functions* on page 19
- *InTouch Functions* on page 40
- *Math Functions* on page 58
- *Miscellaneous Functions* on page 65
- *String Functions* on page 85
- *System Functions* on page 101
- *WWDDE Functions* on page 102

The Type folder contains a set of Microsoft .NET script functions, which are not documented. Refer to Microsoft .NET documentation for descriptions of the functions.

Graphic Client Functions

Use graphic client functions to hide and show symbols, open and close popup windows, log in and log off users, or to query custom properties contained in a symbol.

GetCPQuality()

Returns the Quality value of a custom property. This function is available within any Industrial Graphics client script, but is not supported in InTouch OMI ViewApps (works with InTouch HMI only).

If used in an InTouch OMI ViewApp, GetCPQuality() always returns a value of 0 (bad quality).

Syntax

```
Int          GetCPQuality(String name)
```

Where *String name* is the name of the custom property whose quality is to be retrieved.

This script function takes the name of a custom property on the symbol. This argument is of type string and it can be a reference or a constant.

If the custom property is type constant, GOOD is the quality always returned.

Note: For use with custom properties only. It does not apply to InTouch tags.

Return Value

The GetCPQuality() script function returns a value 0-255 of type Integer, as per the OPC quality standard. 192 is GOOD.

Example

```
cp2 = GetCPQuality("cp1");
```

Where cp1 and cp2 are custom properties and the data type of cp2 is Integer.

GetCPTimeStamp()

Returns the time stamp of a custom property. This function is available within any Industrial Graphics client script.

Syntax

```
DateTime GetCPTimeStamp(String name)
```

Where *String name* is the name of the custom property whose time stamp is to be retrieved.

This script function takes the name of a custom property on the symbol. This argument is of type string and it can be a reference or a constant.

Note: For use with custom properties only. It does not apply to InTouch tags.

Return Value

The GetCPTimeStamp() script function returns the time stamp of the custom property's current value of type DateTime. If the custom property value is a constant, then the return value is the time the value was created.

Example

```
cp2 = GetCPTimeStamp("cp1");
```

Where cp1 and cp2 are custom properties and the data type of cp2 is DateTime.

HideContent()

Closes one or more matching content items within an InTouch OMI ViewApp. Multiple content items can be closed if they match the parameters that are specified in the HideContent call. The HideContent() function uses a subset of the parameters that *ShowContent()* on page 25 uses.

The HideContent() function works only within a single level of the layout, and the level is defined by the SearchScope parameter. By default, SearchScope is "Self," and searches within the layout that has invoked it. This function is available within any Industrial Graphics client script or InTouch OMI layout script. SearchScope parameters other than "Self" constrain the search for content to only layouts that are directly associated with the Screen Profile, and not a nested layout. (A nested layout is a layout embedded or contained in a pane of another layout.)

Note: While the HideContent() function is available in object scripts through both IntelliSense and the IDE function browser, its use in object scripts is not supported.

Category

Graphic Client

Syntax

```
Dim contentInfo as aaContent.ContentInfo;
contentInfo.Content = "SA_Valve_2Way";
contentInfo.Name ="SA_Valve_2Way1";
contentInfo.ContentType = "Level_3";
contentInfo.PaneName ="Pane 2";
contentInfo.ScreenName ="Primary";
contentInfo.SearchScope = aaContent.SearchScope.Self;
```

```
HideContent( contentInfo );
```

Parameter

ContentInfo

The description of the content, along with the location of the content (screen and pane) to be hidden.

Data Type

aaContent.ContentInfo

Examples

```
Dim contentInfo as aaContent.ContentInfo;
contentInfo.Name = "Symbol21";
HideContent( contentInfo );
```

Where "Symbol21" is the Name property of the content shown in the Layout Editor.

```
Dim contentInfo as aaContent.ContentInfo;
contentInfo.Content = "Symbol_001";
HideContent( contentInfo );
```

Where "Symbol_001" is the name of the content as listed in the Graphic Toolbox.

```
Dim contentInfo as aaContent.ContentInfo;
HideContent( contentInfo );
```

When ContentInfo does not define any properties in the HideContent call, the nested layout that called it is hidden. In this case, it works identically to the HideSelf method. See *HideSelf()* on page 25 for more information.

If HideContent() with no ContentInfo properties is called from the top level of a layout, it has no effect; that is, the top level layout is not closed.

Note: Even if you do not define any ContentInfo properties, you must pass the ContentInfo parameter in the HideContent call (i.e., `HideContent ( contentInfo )`).

Property	Definition	Data Type	Required/ Optional
Content	A unique name for an item, either in the Graphic Toolbox or associated with an asset, that specifies the content to be loaded into the pane. Content can be a symbol, a layout, or external content.	String	Optional
Additional information	"Content" is the name of the item within the Graphic Toolbox or associated with an asset. The content names are the names shown in the Graphic Toolbox. The Properties tab of the Layout and ViewApp editors lists content name as the Content property. Relative names, for example, "Me.S1," can also be used to designate content.		

	Content name must be unique. Application Server does not check for duplicated names. If Content is duplicated, all content with the same name is closed. If the same content item is used in multiple panes of the layout, and the "Content" property is specified by the HideContent() method, all instances of the content item are hidden. To hide a single instance of a content item that appears more than once in the layout, use the "Name" property instead.		
Example	`contentInfo.Content = "UserDefinedObject_001.Symbol_001";`		
Name	The auto-generated (or user-edited) name of a unique content item. The name is created when the content item is added to a layout pane.	String	Optional
Additional information	"Name" must be unique within a layout. Name can be duplicated in nested layouts, as long as the name is not duplicated within a nested single layout or the top level layout. The content Name property is shown in the Properties tab of the Layout Editor, and is auto-generated from the Content property, also shown in the Properties tab. You can edit the Name property. When Name is specified in a HideContent call, only the uniquely-named content is closed. If "Content" is specified and the "Name" property is not specified, all items in the layout with the same content name are hidden. If both are specified, "Name" has precedence.		
Example	`contentInfo.Name = "Symbol_011";`		
ScreenName	Specifies the screen that contains the pane with the content to be closed.	String	Optional
Additional information	ScreenNames are configured in the **Screen Profile Editor**. See Screen Profiles in the *System Platform Help* for additional information.		
Example	`contentInfo.ScreenName = "Wall";`		
PaneName	Specifies the pane containing the content to be closed.	String	Optional
Additional information	PaneNames are configured in the **Layout Editor**. See Layouts in the *System Platform Help* for additional information.		
Example	`contentInfo.PaneName = "Pane1";`		
ContentType	Specifies the content type of the content to be closed, for example, "Overview," "Navigation," or "Faceplate."	String	Optional
Additional information	ContentType is matched against the Content Type parameter that can be set for a pane in the Layout Editor. ContentType is used to override the actual type of the specified Content. If ContentType is not specified, Content is examined for its type of content. See Layouts in the *System Platform Help* for additional information about content types.		
Example	`contentInfo.ContentType = "Overview";`		

SearchScope	When ScreenName has not been specified, SearchScope specifies which screen or screens will be searched for a pane that matches the specified PaneName or content type. The default SearchScope is "Self."	Enum	Optional
Additional information	SearchScope is an enum with the following values: • **Self** searches for matching content within the panes of the layout from which the HideContent call was made. If SearchScope is not specified, "Self" is the default. When SearchScope = Self, the layout that initiated the call is searched, whether it is the top level layout or a nested (embedded) layout. In contrast to "Self," the remaining SearchScope values reference only the top level layout, not nested layouts. • **AllScreens** searches for matching content within the panes of all screens in the top level layout. The search starts with the source screen, then the primary screen, and then any remaining screens in alphabetical order. • **SourceScreen** searches for matching content only within the panes of the top level layout from which HideContent was called. • **PrimaryScreen** searches for matching content only within the panes of the top level layout of the screen designated in the Screen Profile as the primary screen.		
Nested Layout	If SearchScope is Self or is not specified, HideContent searches for a matching pane within the layout that initiated the HideContent call. When SearchScope is All, Source, or Primary, or if the ScreenName is specified, HideContent searches for matching content within the top-level layout only.		
Example	`contentInfo.SearchScope = aaContent.SearchScope.PrimaryScreen;`		
Property Overrides	Not applicable for use with HideContent. Used for ShowContent calls only.	NA	NA
Additional information	Property overrides are specified as a key-value pair, with the property name enclosed in quotes. See *ShowContent()* on page 25 for more information.		
OwningObject	Sets the owning object of the content shown by the ShowContent() script function.	NA	NA
Additional Information	OwningObject can be used for relative referencing. Can be a concatenation of constant strings and reference strings. Can be browsed using the Display Automation Object Browser, or you can type the name of the owning object. **Note:** The *OwningObject* property sets references for the graphic, but is not associated with the *GraphicName* property if the symbol is part of an Object Wizard. Therefore, if you are scripting a symbol with an owning object, specify the owning object name as part of the *GraphicName* property, for example, `UserDefined_001.Pump_001`.		

Terms

Content type: specifies the type of content represented by a pane.

Content: the name of a graphic, layout, or external content item as it is listed within the the Graphic Toolbox. This is displayed as the Content property in the Layout Editor when the content item is added to a layout.

Name: the unique name assigned to an instance of a content item, when it is added to a layout. This is displayed as the Name property in the Layout Editor when the content item is added to a layout, and can be edited.

Layout: consists of one or more rectangular areas called panes that contain content shown in a ViewApp. A layout is associated with a screen, or it can be embedded within a pane of another layout.

Embedded or Nested Layout: In the context of ShowContent and HideContent, an embedded layout is a layout that is placed inside a pane of a containing layout. When SearchScope is "Self" (default), embedded layouts are searched for content that matches the parameters specified in the ShowContent/HideContent call.

Pane: rectangular area of a layout that can hold a single piece of content.

Primary screen: represents the main screen of a workstation that will show a running ViewApp

Screen Profile: defines the physical characteristics of one or more client workstation screens that will show a running ViewApp and how these screens are arranged with respect to each other.

Source screen: screen from which ShowContent or HideContent was called.

See Also

ShowContent() on page 25, *HideSelf()* on page 25

HideGraphic()

Closes an open graphic pop-up window shown in the ShowGraphic() script with the given identity name.

The HideGraphic() function has been extended to close InTouch Windows identified with a given identity name. This function is available within any Industrial Graphics client script.

Category

Graphic Client

Syntax

```
HideGraphic(string identity);
```

Parameter

Identity

The unique name of the instance that shows the graphic.

Examples

```
HideGraphic("i1");
```

Where "i1" is string Identity.

```
HideGraphic("InTouch:Window1");
```

Where "InTouch:Window1" is the string identity.

See Also

ShowGraphic() on page 30, *HideSelf()* on page 25

HideSelf()

Closes the displayed graphic or layout for which this script is configured. This script function is available within any Industrial Graphics client script or a nested InTouch OMI Layout script.

Category

Graphic Client

Syntax

```
HideSelf();
```

Remarks

For an Industrial Graphics script, you must call the script function within the symbol to hide the popup. When used in an InTouch OMI Layout script to hide a layout, the script function has no effect on a top level layout; it will only close the layout when called from a nested (embedded) layout.

Example

```
HideSelf();
```

See Also

ShowGraphic() on page 30, *HideGraphic()* on page 24, *ShowContent()* on page 25, *HideContent()* on page 20

Logoff()

Action script that automatically logs off the current user from a ViewApp.

Action scripts are graphic animations that are triggered by a user action such as a mouse click.

Category

Miscellaneous

Syntax

LogOff() ;

Parameter

None

Trigger

On Left-Click/Key/Touch Down

Additional Information

A log off button can be added that uses the Logoff() method to allow the user to log off from the ViewApp.

Example

```
Logoff() ;
```

See Also

ShowLoginDialog() on page 39

ShowContent()

Loads a content item into an InTouch OMI pane. This function is available within an Industrial Graphics client script or layout script to show the content of pane.

To load a graphic into a modal or modeless popup window, use *ShowGraphic()* on page 30. You can use ShowGraphic() for both InTouch OMI and InTouch HMI ViewApps. The ShowContent() method is for InTouch OMI only.

Note: While the ShowContent() function is available in object scripts through both IntelliSense and the IDE function browser, its use in object scripts is not supported.

Category

Graphic Client

Syntax

```
Dim contentInfo as aaContent.ContentInfo;
contentInfo.Content = "SA_Valve_2Way";
contentInfo.Name ="SA_Valve_2Way1";
contentInfo.ContentType = "Level_3";
contentInfo.PaneName ="Pane 2";
contentInfo.ScreenName ="Primary>";
contentInfo.SearchScope = aaContent.SearchScope.Self;
ShowContent( contentInfo );
```

Parameter

ContentInfo

Description of the content to be shown and where to show it (which screen and pane)

Data Type

aaContent.ContentInfo

Example

Show content in a pane

```
dim contentInfo as aaContent.ContentInfo;

Dim cpValues [2] as aaContent.PropertyOverrideValue;
cpValues[1] = new aaContent.PropertyOverrideValue("CP1", "20", true);
cpValues[2] = new aaContent.PropertyOverrideValue("CP2", "Pump.PV.TagName",
false);

contentInfo.Content = "Symbol1";
contentInfo.Name = "S12";
contentInfo.ContentType = "Overview";
contentInfo.OwningObject = "Enterprise";
contentInfo.PaneName = "Pane 1";
contentInfo.ScreenName = "Wall";
contentInfo.PropertyOverrideValues = cpValues;

contentInfo.SearchScope = aaContent.SearchScope.PrimaryScreen;

ShowContent ( contentInfo );
```

aaContent.ContentInfo Properties

ContentInfo is a predefined structure that contains the data members described in the following table.
String properties can be a concatenation of string and/or custom properties.

Note: See "Terms," below, for definitions of Content Type, Layout, Pane, Screen Profile, Primary Screen, and Source Screen.

Property	Definition	Data Type	Required/ Optional
Content	A unique name for an item, either in the Graphic Toolbox or associated with an asset, that specifies the content to be loaded into the pane. Content can be a symbol, a layout, or external content.	String	**Required**
Additional information	"Content" is the name of the item within the Graphic Toolbox or associated with an asset. It can be a symbol, a layout, or external content item. Symbol (graphic), layout, and external content names are listed in the Graphic Toolbox and in the Toolbox tab of the Layout and ViewApp editors. Relative names, for example, "Me.S1," can also be used to designate content. If the specified content is already shown, invoking ShowContent again closes the open content and reopens it. However, if the content is a symbol and you are using object wizards that include Symbol Wizard custom property selections, and the symbol has an owning object, use the symbol's absolute name. This allows the correct symbol configuration to be shown for the instance. See *Owning Object*, below, for more information. Content name must be unique. Application Server does not check for duplicated names. If Content is duplicated, open content with the same name is closed, and the content with the duplicated name is opened in its place. PropertyOverrides and other ContentInfo parameters are updated with any new specified values. If the same content item is used in multiple panes of the layout, and the "Content" property is specified by the HideContent() method, all instances of the content item are hidden. To specify a single instance, use the "Name" property instead.		
Example	`contentInfo.Content = "UserDefinedObject_001.Symbol_001";`		
Name	The auto-generated (or user-edited) name of a unique content item. The name is created when the content item is added to a layout pane.	String	Optional
Additional information	"Name" must be unique within a layout. Name can be duplicated in nested layouts, as long as the name is not duplicated within a nested single layout or the top level layout. If content with the same Name is open within the SearchScope, the matching, open content is closed. A new instance of the matching content opens in the pane specified by the ShowContent call. When Name is specified in a HideContent call, only the uniquely-named content is closed. If "Content" is specified and the "Name" property is not specified, all items in the layout with the same content name are hidden.		

Example	`contentInfo.Name = "Symbol_011";`		
ScreenName	Specifies the screen that contains a pane in which to place the content.	String	Optional
Additional information	ScreenNames are configured in the **Screen Profile Editor**. See Screen Profiles in the *System Platform Help* for additional information.		
Example	`contentInfo.ScreenName = "Wall";`		
PaneName	Specifies the pane in which to place the content.	String	Optional
Additional information	PaneNames are configured in the **Layout Editor**. See Layouts in the *System Platform Help* for additional information.		
Example	`contentInfo.PaneName = "Pane1";`		
ContentType	Specifies the content type, for example, "Overview," "Navigation," or "Faceplate."	String	Optional
Additional information	ContentType is matched against the Content Type parameter that can be set for a pane in the Layout Editor. ContentType is used to override the actual type of the specified Content. If ContentType is not specified, Content is examined for its type of content. See Layouts in the *System Platform Help* for additional information about content types.		
Example	`contentInfo.ContentType = "Overview";`		
SearchScope	When ScreenName has not been specified, SearchScope specifies which screen or screens will be searched for a pane that matches the specified PaneName or content type. The default SearchScope is "Self."	Enum	Optional
Additional information	SearchScope is an enum with the following values: • **Self** searches for matching content within the panes of the NESTED layout (an embedded layout) from which the ShowContent call was made. If SearchScope is not specified, "Self" is the default. In contrast to "Self," the remaining SearchScope values reference only the top level layout, not nested layouts. • **AllScreens** searches for matching content within the panes of all screens in the top level layout. The search starts with the source screen, then the primary screen, and then any remaining screens in alphabetical order. • **SourceScreen** searches for matching content only within the panes of the top level layout from which ShowContent was called. • **PrimaryScreen** searches for matching content only within the panes of the screen designated in the Screen Profile as the primary screen.		
Nested Layout	If SearchScope is Self or is not specified, and ShowContent was called from a nested (embedded) layout, ShowContent searches for a matching pane within the nested layout. When SearchScope is All, Source, or Primary, or if the ScreenName is specified, ShowContent searches for matching content within the top-level layout only.		

Example	`contentInfo.SearchScope = aaContent.SearchScope.PrimaryScreen;`		
Property Overrides	PropertyOverrides sets custom property overrides if a graphic has been specified by the Content property. Each override must include the custom property name and the override value.	Property Override ValuePair[] array	Optional
Additional information	Property overrides are specified as a key-value pair, with the property name enclosed in quotes.		

Example	`Dim cpValues [2] as aaContent.PropertyOverrideValue;`
	`cpValues[1] = new aaContent.PropertyOverrideValue("CP1", "20", true);`
	`cpValues[2] = new aaContent.PropertyOverrideValue("CP2", "Pump.PV.TagName", false);`		

OwningObject	Sets the owning object of the content shown by the ShowContent() script function.	String	Optional
Additional Information	Can be a concatenation of constant strings and reference strings. Can be browsed using the Display Automation Object Browser, or you can type the name of the owning object. **Note:** The *OwningObject* property sets references for the graphic, but is not associated with the *GraphicName* property if the symbol is part of an Object Wizard. Therefore, if you are scripting a symbol with an owning object, specify the owning object name as part of the *GraphicName* property, for example, `UserDefined_001.Pump_001`.		
Example	`contentInfo.OwningObject = "Enterprise";`		

Terms

Content type: specifies the type of content represented by a pane.

Content: the name of a graphic, layout, or external content item as it is listed within the the Graphic Toolbox. This is displayed as the Content property in the Layout Editor when the content item is added to a layout.

Name: the unique name assigned to an instance of a content item, when it is added to a layout. This is displayed as the Name property in the Layout Editor when the content item is added to a layout, and can be edited.

Layout: consists of one or more rectangular areas called panes that contain content shown in a ViewApp. A layout is associated with a screen, or it can be embedded within a pane of another layout.

Embedded or Nested Layout: In the context of ShowContent and HideContent, an embedded layout is a layout that is placed inside a pane of a containing layout. When SearchScope is "Self" (default), embedded layouts are searched for content that matches the parameters specified in the ShowContent/HideContent call.

Pane: rectangular area of a layout that can hold a single piece of content.

Primary screen: represents the main screen of a workstation that will show a running ViewApp

Screen Profile: defines the physical characteristics of one or more client workstation screens that will show a running ViewApp and how these screens are arranged with respect to each other.

Source screen: screen from which ShowContent or HideContent was called.

See Also

HideContent() on page 20

ShowGraphic() on page 30

ShowGraphic()

Shows a graphic within a pop-up window. The ShowGraphic() function has been extended to call InTouch Windows. This function is available within any Industrial Graphics client script.

Category

Graphic Client

Syntax

Show a graphic within a pop-up window

```
Dim graphicInfo as aaGraphic.GraphicInfo;
graphicInfo.Identity = "<Identity>";
graphicInfo.GraphicName = "<SymbolName>";
ShowGraphic( graphicInfo );
```

Call an InTouch window

```
Dim graphicInfo as aaGraphic.GraphicInfo;
graphicInfo0.Identity = "<InTouch:WindowName>";
ShowGraphic( graphicInfo );
```

Parameter

GraphicInfo

Data Type

aaGraphic.GraphicInfo

Examples

Show graphic within a pop-up window

```
ShowGraphic (graphicInfo);
```

Show an InTouch window

```
Dim graphicInfo0 as aaGraphic.GraphicInfo;
graphicInfo0.Identity = "InTouch:Window1";
ShowGraphic( graphicInfo0 );
```

aaGraphic.GraphicInfo Properties

Any string properties can be a concatenation of strings and/or custom properties.

Identity

A unique name that identifies which instance has opened the graphic.

Data Type
String

Additional Information
Mandatory
The same Identity is used in the HideGraphic() script function to close the pop-up window.

Valid Range
The name cannot contain more than 329 characters.
The name must contain at least one letter.
Valid characters are alphanumeric and special characters ($, #, _).

Example

```
graphicInfo.Identity = "i1";
```

GraphicName

The name of the graphic to show.

Data Type
String

Valid Range
The name cannot contain more than 329 characters.
The name must contain at least one letter.
Valid characters are alphanumeric and special characters ($, #, _).

Additional Information
Mandatory
Browse using the **Display Galaxy Browser** or directly type the graphic name.
Galaxy name can come from:

- Graphic Toolbox, for example:
  `"Symbol_001"`
- Instances, absolute or hierarchical, for example:
  `"Userdefined_001.Symbol1", "Userdefined_001.Pump_001.S1"`
- Relative reference, for example:
  `"Me.Symbol_001"`

Use an absolute name to specify the symbol name and owning object if you are using an Object Wizard with Symbol Wizard custom property selections. This allows the correct symbol configuration to be shown for the instance. See *Owning Object*, below, for more information.
If you type any invalid character or exceed the character limit, the system shows a warning message at run time. There is no validation at design time.
The graphic name can be a concatenation of constant strings and reference strings. For example:`"Pump_001" + ".Symbol_001"; cp1 + ".Symbol_001", where the value of cp1 = "Pump_001"; or Obj1.Str1 + ".Symbol_001", where the value of Obj.Str1 = "Pump_001".`

Examples

Graphic Toolbox Reference

```
graphicInfo.GraphicName = "S1";
```

Absolute Reference

```
graphicInfo.GraphicName = "OwningObjectName.SymbolName";
```

OwningObject

The owning object of the graphic shown by the ShowGraphic() script function.

Data Type
String

Default Value
Empty

Additional Information
Optional
Can be a concatenation of constant strings and reference strings.
Can be browsed using the **Display Automation Object Browser**, or you can type the name of the owning object.

Note: The *OwningObject* property sets references for the graphic, but is not associated with the *GraphicName* property if the symbol is part of an Object Wizard. Therefore, if you are scripting a symbol with an owning object, specify the owning object name as part of the *GraphicName* property, for example, `UserDefined_001.Pump_001`.

Example

```
graphicInfo.OwningObject = "UserDefined_001";
```

HasTitleBar

Determines if the graphic is shown with a title bar.

Data Type
Boolean

Default Value
True

Example

```
graphicInfo.HasTitleBar = false;
```

WindowTitle

Specifies the title shown in the window title bar.

Data Type
String

Default Value
Empty

Valid Range
Limit 1024 characters

Additional Information
Can be a constant string, a reference, or an expression.
If you change the owning object for an AutomationObject graphic, the window title is updated accordingly.
If the WindowTitle parameter is empty, the value of the Identity parameter is shown on the title bar.

Example

```
graphicInfo.WindowTitle = "Graphic01";
```

WindowType

Specifies whether window type is modal or modeless.

Data Type
Enum

Default Value
Modeless

Valid Range
0, 1

Enumerations

WindowType	Integer
Modal	0
Modeless	1

Examples

```
graphicInfo.WindowType = aaGraphic.WindowType.<windowtype>;
graphicInfo.WindowType = 1;
```

HasCloseButton

Determines if the pop-up window has a close button.

Data Type
Boolean

Default Value
True

Example

```
graphicInfo.HasCloseButton = false;
```

Resizable

Determines if the pop-up window is resizable.

Data Type
Boolean

Default Value
False

Example

```
graphicInfo.Resizable = true;
```

WindowLocation

Specifies the location of the pop-up window.

Data Type
Enum

Default Value
Center

Valid Range
One of 0–12

Enumerations

WindowLocation	Integer
Center	0
Above	1
TopLeftCorner	2
Top	3
TopRightCorner	4
LeftOf	5
LeftSide	6
RightSide	7
RightOf	8

WindowLocation	Integer
BottomLeftCorner	9
Bottom	10
BottomRightCorner	11
Below	12

Additional Information
If you have selected Desktop as the window relative position, Above, LeftOf, RightOf, and Below are invalid.
For more information about the behavior of the WindowLocation parameter, see "Working with the Show/Hide Graphics Script Functions," in the *Creating and Managing Industrial Graphics User Guide.*

Examples

```
graphicInfo.WindowLocation = aaGraphic.WindowLocation.<WindowLocation>;
graphicInfo.WindowLocation = 1;
```

WindowRelativePosition

Specifies the relative position of the pop-up window.

Data Type
Enum

Default Value
Desktop

Valid Range
One of 0–8

Enumerations

WindowRelativePosition	Integer
Desktop	0
Window	1
ClientArea	2
ParentGraphic	3
ParentElement	4
Mouse	5
DesktopXY	6
WindowXY	7
ClientAreaXY	8

Examples

```
graphicInfo.WindowRelativePosition =
aaGraphic.WindowRelativePosition.<WindowRelativePosition>;
graphicInfo.WindowRelativePosition = 1;
```

RelativeTo

Specifies the size of the pop-up window relative to the graphic, desktop, or customized width and height.

Data Type
Enum

Default Value
Graphic

Valid Range
One of 0–2

Enumerations

RelativeTo	Integer
Graphic	0
DeskTop	1
CustomizedWidthHeight	2

Additional Information
If you enter aaGraphic.RelativeTo.CustomizedWidthHeight, you can include the values of the height and width in the script. Otherwise, the default values are used.

Examples
```
graphicInfo.RelativeTo = aaGraphic.RelativeTo.<RelativeTo>;
graphicInfo.RelativeTo = 1;
```

X

The horizontal position of the pop-up window.

Data Type
Integer

Default Value
0

Valid Range
-2,147,483,648 through 2,147,483,647

Additional Information
If X is beyond the integer range, an overflow message appears in the Logger at run time.
This parameter is applicable only if the value of the WindowRelativePosition parameter is DesktopXY, WindowXY, or ClientAreaXY.
Unlike the ShowSymbol animation, there is no boundary for this value.

Examples
```
graphicInfo.X = 100;
```

Y

Specifies the vertical position of the pop-up window.

Data Type
Integer

Default Value
0

Valid Range
-2,147,483,648 through 2,147,483,647

Additional Information
If Y is beyond integer range, a proper overflow message will appear in the Logger at run time.
This value is applicable only if WindowRelativePosition is DesktopXY, WindowXY, or ClientAreaXY.

Unlike the ShowSymbol animation, there is no boundary for this value.

Examples
```
graphicInfo.Y = 100;
```

Width

Specifies the width of the pop-up window.

Data Type
Integer

Default Value
100

Valid Range
0–10000

Additional Information
Applicable only if RelativeTo is CustomizedWidthHeight
You can specify either the height or the width of the pop-up window. The system calculates the other, based on the aspect ratio of the symbol.
If you enter an out-of-boundary value, the system shows an "Out of range" message at run time. If the value > 10000, it is set at 10000. If the value < 0, it is set at 0.

Examples
```
graphicInfo.width = 500;
```

Height

Specifies the height of the pop-up window.

Data Type
Integer

Default Value
100

Valid Range
0–10000

Additional Information
Applicable only if RelativeTo is the value of the CustomizedWidthHeight parameter.
You can specify either the height or the width of the pop-up window. The system calculates the other, based on the aspect ratio of the symbol.
If you enter an out-of-boundary value, the system shows an "Out of range" message at run time. If the value > 10000, it is set at 10000. If the value < 0, it is set at 0.

Examples
```
graphicInfo.height = 500;
```

TopMost

Sets a value that indicates whether the ShowGraphic appears in the top most z-order window. A ShowGraphic whose Topmost property is set to true appears above all windows whose TopMost properties are set to false (same as Windows Task Manager).

Data Type
Boolean

Default Value
False

Additional Information
ShowGraphic windows whose Topmost properties are set to true appear above all windows whose Topmost properties are set to false. In a group of windows that have the Topmost property set to true, the active window is the topmost window.

Note: Do not create scripts that launch a non-TopMost Modal dialog from a TopMost dialog. Users will not be able to interact with the View if the Modal dialog is completely hidden by any TopMost window.

Example
```
graphicInfo.TopMost = true;
```

ScalePercentage

Sets the scaling percentage of the pop-up window and the graphic it contains.

Data Type
Integer

Default Value
100

Valid Range
0–1000

Additional Information
If you enter an out-of-boundary value, the system shows an "Out of range" message at run time. If the value > 1000, it is set at 1000. If the value < 0, it is set at 0.

Examples
```
graphicInfo.ScalePercentage = 150;
```

KeepOnMonitor

Specifies that a pop-up window should appear entirely within the boundaries of an application window.

Data Type
Boolean

Default Value
True

Example
```
graphicInfo.KeepOnMonitor = true;
```

StretchGraphicToFitWindowSize

Determines if the graphic is scaled to the current size of the pop-up window.

Data Type
Boolean

Default Value
True

Additional Information
Applicable only if the value of the ScalePercentage parameter is greater than 100.

Examples
```
graphicInfo.StretchGraphicToFitWindowSize = false;
```

StretchWindowToScreenWidth

Determines if the pop-up window is scaled to the same width as the screen.

Data Type
Boolean

Default Value
False

Additional Information
Applicable only if the WindowRelativePosition parameter is Desktop, Window, Client Area, ParentGraphic, or ParentElement.

Examples
```
graphicInfo.StretchWindowToScreenWidth = true;
```

StretchWindowToScreenHeight

Determines if the pop-up window is scaled to the same height as the screen.

Data Type
Boolean

Default Value
False

Additional Information
Applicable only if the WindowRelativePosition parameter is Desktop, Window, Client Area, ParentGraphic, or ParentElement.

Examples
```
graphicInfo.StretchWindowToScreenHeight = true;
```

CustomProperties

Sets the custom properties of the symbol being shown.

Data Type
CustomPropertyValuePair[] array

Additional Information
The first three parameters are custom property name, value, and IsConstant.
Both custom property and the value can be a constant string, reference, or concatenation of strings. If the parameter IsConstant = True, the value is treated as a constant. Otherwise, the value is treated as a reference.
The array index starts at 1.

Examples
```
Dim cpValues [4] as aaGraphic.CustomPropertyValuePair;
cpValues[1] = new aaGraphic.CustomPropertyValuePair("CP1", 20, true);
cpValues[2] = new aaGraphic.CustomPropertyValuePair("CP2", Pump.PV.TagName,
true);
cpValues[3] = new aaGraphic.CustomPropertyValuePair("CP3", "CP"+var1, CP2 +
"001" + ".Speed", true);
cpValues[4] = new aaGraphic.CustomPropertyValuePair("CP3", "InTouch:Tag1",
false);
graphicInfo.CustomProperties = cpValues;
```

Remarks

Any parameter that has default value in the GraphicInfo is optional. If no input value specified for these parameters, the default values are used at run time. Any parameter except the Enum data type can be a constant, reference, or expression.

For more information, see "Working with the Show/Hide Graphics Script Functions" in the *Creating and Managing Industrial Graphics User Guide*.

Examples for ShowGraphic

Basic script example:

```
Dim graphicInfo as aaGraphic.GraphicInfo;
graphicInfo.Identity = "Script_001";
graphicInfo.GraphicName = "Symbol_001";
ShowGraphic( graphicInfo );
```

Advanced script example:

```
Dim graphicInfo as aaGraphic.GraphicInfo;
Dim cpValues [2] as aaGraphic.CustomPropertyValuePair;
cpValues[1] = new aaGraphic.CustomPropertyValuePair("CP1", 20, true);
cpValues[2] = new aaGraphic.CustomPropertyValuePair("CP2", "Pump.PV.TagName",
false);
graphicInfo.Identity = "i1";
graphicInfo.GraphicName = "S1";
graphicInfo.OwningObject = "UserDefined_001";
graphicInfo.WindowTitle = "Graphic01";
graphicInfo.Resizable = false;
graphicInfo.CustomProperties=cpValues;
ShowGraphic( graphicInfo );
```

Where "i1" is string Identity and the symbol "S1" contains custom property CP1 and CP2.

See Also

HideSelf() on page 25

ShowLoginDialog()

Action script that shows a login dialog box with fields to enter a username and password. A typical login interface includes a login button that is selected by the user to show the **Login** dialog box with fields to enter a username and password.

Action scripts are graphic animations that are triggered by a user action such as a mouse click.

Category

Miscellaneous

Syntax

ShowLoginDialog() ;

Parameter

None

Trigger

On Left-Click/Key/Touch Down

Additional Information

A log off button can be added that uses the Logoff() method to allow the user to log off from the ViewApp.

Example

```
ShowLoginDialog() ;
```

See Also

Logoff() on page 25

InTouch Functions

The following InTouch functions can be used within the script editor contained in the **Industrial Graphic Editor**. In all functions that specify tag names as parameters, you can use InTouch tags from your InTouch application.

InTouch script functions can be used *only* in symbol scripts. InTouch script functions do not work in ArchestrA object scripts. Even though the object script will not work, no error or warning is generated.

Note: Using the **Convert to Industrial Graphic** option in InTouch scripts may wrongly append the term *"InTouch:"* to the script function name. To avoid errors, remove the term *"InTouch:"* from the script function name.

AddPermission() Function

Assigns a certain InTouch access level to a given user group on the local system or on the domain. When a user belonging to that group logs on to the InTouch HMI after the AddPermission() function is called, he or she receives the specified access level.

Category

security

Syntax

```
DiscreteTag=AddPermission( "Domain", "Group", AccessLevel);
```

Arguments

Domain
Name of the domain or local computer in which the group is located.

Group
Windows user group.

AccessLevel
InTouch access level that you want to associate with the given group.

Remarks

Valid for operating system security only. When this function is called, it checks for the presence of the specified group in the specified domain or workgroup. If successful, TRUE is returned, and the specified Access Level is associated with the group for subsequent user log ons. In all other cases, (that is, if an invalid value is specified for any of the arguments) FALSE is returned.

This function is typically configured to run on application startup. It does not affect users that are currently logged on. Only users that log on after AddPermission() is successfully called receive the access level associated with their group.

Examples

```
DiscreteTag=AddPermission( "corporate_hq", "InTouchAdmins", 9000);
DiscreteTag=AddPermission( "johns01", "InTouchUsers", 5000);
```

See Also

PostLogonDialog(), InvisibleVerifyCredentials(), IsAssignedRole(), AttemptInvisibleLogon(), QueryGroupMembership()

AttemptInvisibleLogon() Function

The AttemptInvisibleLogon() function can be used in a script to log on a user to InTouch using the supplied credentials. The user is not required to enter a password or user ID.

Category

security

Syntax

```
DiscreteTag=AttemptInvisibleLogon( "UserId", "Password", "Domain" );
```

Arguments

UserId
A valid user account name.

Password
Password of the user.

Domain
Name of the local computer, workgroup, or domain to which the user belongs. This argument applies only if the current security type is operating system-based.

Return Value

Returns TRUE if authentication is successful. Otherwise, it returns FALSE.

Remarks

An attempt is made to log on to the InTouch HMI using the supplied credentials.

- If the logon attempt succeeds, then TRUE is returned and the $OperatorDomain, $OperatorName, $AccessLevel, and $Operator system tags are updated accordingly.
- If the log on attempt fails, then FALSE is returned, and the currently logged on user (if any) continues to be the current user.

The *Domain* argument is only valid for operating system-based security. If ArchestrA security mode is in use and if ArchestrA security is in turn using operating system-based security, the *UserId* argument should contain the fully qualified user name with domain name or computer name.

Examples

When security is operating system-based:

```
DiscreteTag=AttemptInvisibleLogon("UserId", "Password", "Domain" );
```

When security is either InTouch-based or ArchestrA-based:

```
DiscreteTag=AttemptInvisibleLogon("UserId", "Password", "" );
```

See Also

PostLogonDialog(), InvisibleVerifyCredentials(), IsAssignedRole(), QueryGroupMembership(), AddPermission()

ChangePassword() Function

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

Category

security

Syntax

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

Return Value

Returns one of the following integer values:

0 = Cancel was pressed.

1 = OK was pressed.

Remarks

If the operator uses a touch screen, the operator can use the alphanumeric keyboard to enter the new password.

Example

The following script can be placed on a button or called from a condition script or data change script.

```
Errmsg=ChangePassword();
```

EnableDisableKeys() Function

Enables/disables key filters for the Alt, Escape, and Windows keys.

Category

View

Syntax

```
EnableDisableKeys(AltKey, EscKey, WinKey);
```

Parameters

AltKey
Integer to enable or disable key filters for the Alt key:

1 = enable filter (disable Alt key)

0 = disable filter (enable Alt key)

EscKey
Integer to enable or disable key filters for the Escape key:

1 = enable filter (disable Esc key)

0 = disable filter (enable Esc key)

WinKey
Integer to enable or disable key filters for the Windows key:

1 = enable filter (disable Win key)

0 = disable filter (enable Win key)

Remarks

Disabling the Alt key also disables the Win+L key combination (for locking the Windows desktop). Win+L is the shortcut for another combination of keys that involves the Alt key. Thus, disabling the Alt key also disables the shortcut for locking the Windows desktop.

Disabling the Esc key disables it for all actions.

Example(s)

```
EnableDisableKeys(0,0,0); // enable all three keys
EnableDisableKeys(1,1,1); // disable all three keys
EnableDisableKeys(0,0,1); // enable Alt and Escape keys, disable Windows key.
```

FileCopy() Function

Copies a source file to a destination file and returns a status result. This function may take a longer time to execute and is executed in multiple stages:

1. FileCopy() function is called and an immediate result is returned, indicating success or failure of the file copy initialization.

2. FileCopy() function executes the copy procedure in the background, and InTouch scripting continues execution while the file copying is in progress. You can monitor the file copying progress with an integer tag.
3. FileCopy() function returns a file copy result, indicating success or failure of the file copy procedure.

If the destination folder is not available (i.e. another computer on the network), the function waits for up to 10 seconds to time out, and then posts a message in the Logger.

Note: Do not use the FileCopy() function in asynchronous QuickFunctions.

Syntax

```
result = FileCopy (sourcefile, destfile, progresstag)
```

Parameters

sourcefile
Full path and file name of the file to be copied. A literal string value, message tagname, or string expression. You can use the wildcard characters (* and ?) in this parameter to copy just files matching a specified criteria. The path name can also be a UNC path name.

destfile
Full path and file name (or just path name) of the destination. A literal string value, message tagname, or string expression. The path name can also be a UNC path.

progresstag
Name of an integer tag enclosed in double quotes that will contain a value indicating the file copy progress. A literal string value, message tagname (such as a message tag containing the value "IntTag.Name") or string expression. The values have following meaning:
0 - FileCopy() procedure is still in progress.
1 - FileCopy() procedure has completed successfully.
-1 - FileCopy() procedure completed with errors.

Return Value

A value of -1, 0, or 1 indicating the following:
1 - FileCopy() function successfully called.
0 - Error when calling the FileCopy() function because another FileCopy() procedure is already in progress.
-1 - Error when calling the FileCopy() function because of a non-existent source file or the destination is read only.

Example(s)

This script copies the file c:\MyData\output.log to the directory d:\archive and renames the file to output.txt. The progress of the file copy is written to the integer tag Monitor.

```
Status=FileCopy("c:\MyData\output.log","d:\archive\output.txt","Monitor");
```

This script copies all files with file ending .txt in the c:\ root directory to the destination directory c:\Backup.

```
Status=FileCopy("c:\*.txt", "c:\Backup", "Monitor");
```

This script copies a file whose full path and file name is contained in the message tag LogFile to the destination directory c:\results\ and renames it to logxxx.txt where xxx is a timestamp.

```
Status=FileCopy(LogFile, "c:\results\log" + $DateString + $TimeString + ".txt",
"Monitor");
```

FileDelete() Function

Deletes an individual file.

Syntax

```
result = FileDelete (filename)
```

Parameters

filename
The path name and file name of the file to delete. A literal string value, message tagname, or string expression. UNC path names are supported.

Remarks

Do not use the wildcard characters (* and ?) with the FileDelete() function and do not use the FileDelete() function in asynchronous QuickFunctions.

The FileDelete() function does not delete directories.

Return Value

A value indicating success or failure of the file deletion:
1 - file is deleted successfully
0 - file is not deleted successfully. Possible causes are attempts to delete a read only or a non-existent file.

Example(s)

This script deletes the file c:\Data.txt and returns 1 if the file was found and deleted successfully.

```
Status=FileDelete("c:\Data.txt");
```

FileMove() Function

Moves a source file to a destination file and returns a status result. It can be also used to rename a file. This function may take a longer time to execute and executes in multiple stages:

1. FileMove() function is called and an immediate result is returned, indicating success or failure of the file move initialization.
2. FileMove() function executes the move procedure in the background, InTouch scripting continues execution while the file moving is in progress. You can monitor the file moving progress with an integer tag.
3. FileMove() function returns a file move result, indicating success or failure of the file moving procedure.

Do not use the FileMove() function in asynchronous QuickFunctions.

Syntax

```
result = FileMove (sourcefile, destfile, progresstag)
```

Parameters

sourcefile
Full path and file name of the file to be moved. A literal string value, message tagname, or string expression. You can use the wildcard characters (* and ?) in this parameter to move just files matching a specified criteria. The path name can also be a UNC path name.

destfile
Full path and file name (or just path name) of the destination. A literal string value, message tagname, or string expression. The path name can also be a UNC path.

progresstag
Name of an integer tag enclosed in double quotes that will contain a value indicating the file moving progress. A literal string value, message tagname (such as a message tag containing the value "IntTag") or string expression. The values have following meaning:

0 - FileMove() procedure is still in progress

1 - FileMove() procedure has completed successfully

-1 - FileMove() procedure completed with errors

Return Value

A value of-1, 0, or 1 indicating the following:

1 - FileMove() function successfully called

0 - Error when calling the FileMove() function because another FileMove() procedure is already in progress

-1 - Error when calling the FileMove() function. Possible errors are attempts to move a non-existent file.

Example(s)

This script moves the file c:\MyData\output.log to the directory d:\archive and renames the file to output.txt. The progress of the file moving is written to the integer tag Monitor.

```
Status=FileMove("c:\MyData\output.log","d:\archive\output.txt","Monitor");
```

This script moves all files with file ending .txt in the c:\ root directory to the destination directory c:\Backup.

```
Status=FileMove("c:\*.txt", "c:\Backup", "Monitor");
```

This script moves a file whose full path and file name is contained in the message tag LogFile to the destination directory c:\results\ and renames it to logxxx.txt where xxx is a timestamp.

```
Status=FileMove(LogFile, "c:\results\log" + $DateString + $TimeString + ".txt",
"Monitor");
```

FileReadFields() Function

Reads the values contained in a csv file into a series of tagnames. You can use this function to load a set of tagname values.

Commas are the only supported delimiter.

This function can only be used for synchronous calls.

Syntax

```
[result = ] FileReadFields (filename, offset, starttag, numberoffields)
```

Parameters

filename
Name of the csv file to read the data from. A literal string value, a message tagname or a string expression.

offset
Location (in bytes) in the file to start reading. A literal integer value, integer tagname, or integer expression.

starttag
Name of the first tagname that receives the first read data item. The tagname must be enclosed with double quotes and end in a number, such as "MyTag1". A literal string value, message tagname (such as a message tagname containing the value "MyTag1"), or a string expression.

numberoffields
Number of data items to read from the csv file. A literal integer value, integer tagname, or integer expression. The first data item is read into the tagname defined in the starttag parameter, subsequent data items into tagnames with the incremented numeral suffix of the starttag parameter (MyTag1, MyTag2, MyTag3, ...).

Return Value

Optional new file offset (in byte) after reading the data. This can be used to read the next set of data.

Example(s)

This script reads the values "Flour" to RecipeTag1, 27.23 to RecipeTag2, 14 to RecipeTag3, and 1 to RecipeTag4, and returns the new file offset—if the csv file c:\set.csv contains the following data: Flour, 27.23,14,1 and if the following tags are defined: RecipeTag1:message, RecipeTag2:real, Recipe3:integer, RecipeTag4:discrete.

```
FileReadFields("c:\set.csv",0,"RecipeTag1",4);
```

FileReadMessage() Function

Reads a specified number of bytes (or one line) of string data from a file.

Syntax

```
[result = ] FileReadMessage (filename, offset, messagetag, charstoread)
```

Parameters

filename
Name of the file to read the data from. A literal string value, a message tagname, or a string expression.

offset
Location (in bytes) in the file to start reading from. A literal integer value, integer tagname, or integer expression.

messagetag
Message tagname that receives the first line or number of bytes from the file. Enclose the tagname with double quotes when using the function within the Industrial Graphics Editor Script Editor.

charstoread
Number of bytes to read from the file. Set it to 0 to read until the next line feed (LF) character. A literal integer value, integer tagname, or integer expression.

Return Value

Contains the new byte position after the read. You can use this for subsequent reads from the file.

Example(s)

This script reads the first line of data in the file c:\Data\File.txt to the message tagname MsgTag.

```
FileReadMessage ("c:\Data\File.txt",0,MsgTag, 0);
FileReadMessage ("c:\Data\File.txt",0,"InTouch:MsgTag", 0);
```

FileWriteFields() Function

Writes the values contained in a series of tagnames to a csv file. You can use this function to save a set of tagname values.

Commas are the only supported delimiter.

Syntax

```
[result = ] FileWriteFields (filename, offset, starttag, numberoffields)
```

Parameters

filename
Name of the csv file to write the data to. A new file is created if it does not previously exist. A literal string value, a message tagname, or a string expression.

offset
Location (in bytes) in the file to start writing to. Use -1 to write to the end of the file (append). A literal integer value, integer tagname, or integer expression.

starttag
Name of the first tagname that contains the first data item to be written. The tagname must be enclosed with double quotes and end in a number, such as "MyTag1". A literal string value, message tagname (such as a message tagname containing the value "MyTag1") or a string expression.

numberoffields
Number of data items to write to the csv file. A literal integer value, integer tagname, or integer expression. The first data item is written from the tagname defined in the starttag parameter to the file, subsequent data items from tagnames with the incremented numeral suffix of the starttag parameter (MyTag1, MyTag2, MyTag3, ...).

Return Value

Optional new file offset (in byte) after writing the data. This can be used to write the next set of data.

Example(s)

A series of InTouch tags is defined as follows:

Tagname	Data Type	Value
RecipeTag1	Message	Flour
RecipeTag2	Real	27.23
RecipeTag3	Integer	14
RecipeTag4	Discrete	1

This script writes the values contained in RecipeTag1 to RecipeTag4 to the csv file c:\set.csv.

```
FileWriteFields("c:\set.csv",0,"RecipeTag1",4);
```

So that the file c:\set.csv will contain the following data:

```
Flour,27.23,14,1
```

FileWriteMessage() Function

Writes a specified number of bytes (or one line) of string data to a file.

Syntax

```
[result = ] FileWriteMessage (filename, offset, messagetag, linefeed)
```

Parameters

filename
Name of the file to write the data to. A literal string value, a message tagname, or a string expression.

offset
Location (in bytes) in the file to start writing to. Set it to -1 to write data to the end of the file (append). A literal integer value, integer tagname, or integer expression.

messagetag
Message tagname that contains the data to be written to the file.v

linefeed
Specifies whether to write a line feed (LF) character after writing the data to the file. Set to 1 to write a line feed character; otherwise, set it to 0. A literal Boolean value, discrete tagname, or Boolean expression.

Return Value

Contains the new byte position after the write. You can use this for subsequent writes to the file.

Example(s)

This script writes the value of a message tagname MsgTag to the end of the file c:\Data\File.txt.

```
FileWriteMessage("c:\Data\File.txt",-1,MsgTag,1);
```

GetAccountStatus() Function

Returns the number of days until the user's password expires.

Category

security

Syntax

```
Result=GetAccountStatus(Domain, UserID);
```

Arguments

Domain
Name of the domain or local computer in which the user account is located.

UserID
Windows user account name that is part of the local computer, workgroup, or domain.

Return Value

This function also returns the following values:

Result	Description
-1	Account password expired
-2	Account password never expires
-3	Account locked out
-4	Account disabled
-5	Account info failed

Remarks

Use this script function with operating system-based security. Do not use this function with the ArchestrA security mode.

If the GetAccountStatus() function is used with ArchestrA security, the script attempts to retrieve the account information directly from the domain controller. This works as long as the ArchestrA Galaxy Repository is using operating system security with the same domain.

Example(s)

```
Status = GetAccountStatus("Corporate_HQ","Operator");
```

GetNodeName() Function

Returns the node name of the computer.

Syntax

```
GetNodeName (messagetag, nodenum);
```

Parameters

messagetag
Message tagname that will contain the node name. Enclose the tagname with double quotes when using the function within the Industrial Graphics Editor Script Editor.

nodenum
Number of characters to retrieve from the node name. A literal integer value, integer tagname, or integer expression in the range of 0 to 131.

Example(s)

This script retrieves the node name and assigns it to the NodeName message tagname.

```
GetNodeName(NodeName,131);
GetNodeName("InTouch:NodeName",131);
```

InfoAppTitle() Function

Returns the application title or Windows task list name of a specified application that is running.

Syntax

```
result = InfoAppTitle (appname)
```

Parameters

appname
Name of the application without the .exe extension. A literal string value, message tagname, or string expression.

Example(s)

This script returns "Calculator"

```
InfoAppTitle("calc")
```

This script returns "Microsoft Excel"

```
InfoAppTitle("excel")
```

InfoDisk() Function

Returns either the total or free space on a local or network disk drive.

Syntax

```
result = InfoDisk (drive, infotype, trigger);
```

Parameters

drive
The drive letter for which you want to retrieve information. Only the first character of a string is used. A literal string value, message tagname, string expression.

infotype
Specifies the information type. A literal integer value, integer tagname, or integer expression with following possible values:

1 - function returns total size of disk drive (in bytes)

2 - function returns free space of disk drive (in bytes)

3 - function returns total size of disk drive (in kilobytes)

4 - function returns free space of disk drive (in kilobytes)

trigger
A tagname (or expression) that acts as a trigger to recalculate the disk information. If the trigger value changes the disk information is recalculated. A discrete or analog taname, or a discrete or analog expression.

Remarks

The trigger tag only has meaning when the InfoDisk() function is used in an animation display link. If this function is used in a script, you can specify any literal numeric value, analog tagname, or numeric expression.

Example(s)

Use this script in an animation display link to show the free space of disk drive C and update the information every minute.

```
InfoDisk("C", 4, $Minute)
```

InfoFile() Function

Returns various information on a file or directory.

Syntax

```
result = InfoFile (filename, infotype, trigger)
```

Parameters

filename
The full file name or directory name you want to retrieve information about. A literal string value, message tagname, or string expression. Can also include wildcard characters, such as "*" and "?".

infotype
The type of information you want to retrieve about the specified file or directory. A literal integer value, integer tagname, or integer expression with following values and meaning:

1 - Existence. The InfoFile() function returns 1 if the file exists, 2 if the file is a directory and 0 if the file or directory does not exist.

2 - Size. The InfoFile() function returns the file size in bytes.

3 - Creation timestamp. The InfoFile() function returns the time stamp as seconds that have passed since midnight January 1st 1970. Use the StringFromTimeLocal() function to convert this value to a message timestamp.

4 - Wildcard Search Match. The InfoFile() function returns the number of files that match a specified wildcard search.

trigger
A tagname (or expression) that acts as a trigger to recalculate the file information. If the trigger value changes, the file information is recalculated. A discrete or analog taname, or a discrete or analog expression.

Remarks

The trigger tag only has meaning when the InfoFile() function is used in an animation display link. If this function is used in a script, you can specify any literal numeric value, analog tagname, or numeric expression.

Example(s)

This script returns 1 if the file c:\data\log.txt exists.

```
InfoFile("c:\data\log.txt",1,$minute)
```

This script returns 14223 if the file c:\data\log.txt has a file size of 14223 bytes.

```
InfoFile("c:\data\log.txt",2,$minute)
```

This script returns 1138245266 if the file c:\data\log.txt was created on 26th January 2006 at 11:14:26 AM.

```
InfoFile("c:\data\log.txt",3,$minute)
```

This script returns 14 if there are 14 files in the directory c:\data\ that have a txt ending.

```
InfoFile("c:\data\*.txt",4,$minute)
```

InfoInTouchAppDir() Function

Returns the current InTouch application directory.

Syntax

```
result = InfoInTouchAppDir();
```

Return Value

A message tagname to contain the directory of the currently running InTouch application.

Remarks

The application directory name may be truncated when passed to a message tagname or shown in an animation link due to the 131 characters limitation.

Example(s)

This script may return c:\documents and settings\user1\my documents\my intouch applications\packaging.

```
InfoInTouchAppDir()
```

InTouchVersion() Function

Returns the complete InTouch version number or just parts of it.

Syntax

```
result = InTouchVersion (infotype);
```

Parameters

infotype
Specifies how the version information is returned. A literal integer value, integer tagname, or integer expression with the following meaning:

0- function returns the whole version number

1- function returns just the major version number

2- function returns just the minor version number

3- function returns just the patch level

4- function returns just the build level

Example(s)

Function	Possible result
InTouchVersion(0)	10.5.1626.0521.0045.0012
InTouchVersion(1)	10
InTouchVersion(2)	5
InTouchVersion(3)	0
InTouchVersion(4)	1626

InvisibleVerifyCredentials() Function

The InvisibleVerifyCredentials() function can be used in a synchronous QuickScript to verify the credentials of the given user without logging the user on to the InTouch HMI.

Category

security

Syntax

```
AnalogTag=InvisibleVerifyCredentials( "UserId", "Password", "Domain" );
```

Arguments

UserId
Windows operating system user account name that is part of local computer, workgroup, or domain.

Password
Password for the account.

Domain
The Windows domain for the account.

Remarks

If the supplied combination of user, password, and domain are valid then the corresponding access level associated with the user is returned as an integer. Otherwise, -1 is returned.

Note: The InvisibleVerifyCredentials() function must be run from a synchronous QuickScript. The function always returns -1 if run from an asynchronous QuickScript.

This function does not change the currently logged on user. The Domain argument is only valid for operating system-based security. If ArchestrA security is in use and if ArchestrA security is in turn using operating system-based security, the UserId argument should contain the fully qualified user name with domain name or computer name.

Example

```
AnalogTag=InvisibleVerifyCredentials( "john", "Password", "corporate_hq" );
```

See Also

PostLogonDialog(), AttemptInvisibleLogon(), IsAssignedRole(), QueryGroupMembership(), AddPermission()

IsAssignedRole() Function

Determines whether the currently logged on user is a member of the specified user role. Only applies to ArchestrA security.

Category

security

Syntax

```
DiscreteTag=IsAssignedRole( "RoleName" );
```

Arguments

RoleName
The role associated with an Application Server user.

Remarks

Valid for ArchestrA security mode only and applies to the currently logged on user. If a user is currently logged on and has the *RoleName* role assigned in the Galaxy IDE, then TRUE is returned. Otherwise, FALSE is returned.

Example

```
DiscreteTag=IsAssignedRole( "Administrators" );
```

See Also

AttemptInvisibleLogon(), PostLogonDialog(), InvisibleVerifyCredentials(), QueryGroupMembership(), AddPermission()

LaunchTagViewer() Function

You can start Tag Viewer only when WindowViewer is running, and only after Tag Viewer has been enabled in WindowMaker.

For information about enabling Tag Viewer, see Configuring General WindowViewer Properties in the *InTouch® HMI Application Management and Extension Guide.*

Syntax

```
LaunchTagViewer()
```

Remarks

The LaunchTagViewer() function can be executed from any script type except the application scripts OnStartup and OnShutdown.

If Tag Viewer has not been enabled in WindowMaker, calling the function will not start Tag Viewer and a warning message will appear in the logger.

You must have adequate security privileges to start Tag Viewer.

LogonCurrentUser() Function

Logs on to InTouch with a user account that is currently logged on to the Windows operating system.

- InTouch configured with OS security: the user is logged on to WindowViewer.
- InTouch configured with ArchestrA security: the user must be a member of ArchestrA OS user-based or OS group-based security.
- InTouch configured with ArchestrA OS user-based or OS group-based security and the user account is configured with smart card credentials: user is logged on using the smart card credentials. The user is logged off if the smart card is removed from the reader.

Category

security

Syntax

```
IntegerResult = LogonCurrentUser();
```

Return Value

Returns -1 and no change to the values assigned to $Operator, $OperatorName, $OperatorDomain, and $AccessLevel if the logon fails.

Remarks

This function is available only in InTouch scripting, not in ArchestrA client scripting.

Example

```
IntegerResult = LogonCurrentUser();
```

See Also

PostLogonDialog(), InvisibleVerifyCredentials(), IsAssignedRole(), AttemptInvisibleLogon(), QueryGroupMembership(), AddPermission()

PlaySound() Function

Plays a sound from a wave file or a Windows default sound.

Syntax

```
Playsound (soundname, flag)
```

Parameters

soundname
The name of the sound or wave file. A literal string value, message tagname, or string expression. If the sound is defined as a name, it must be defined in the Win.ini file under the [Sounds] section, for example MC="c:\test.wav"

flag
Specifies how the sound is played. A literal integer value, integer tagname, or integer expression with the following meanings:
0 - Play sound one time synchronously (script execution waits until sound has finished playing).
1 - Play sound one time asynchronously (script execution does not wait until sound has finished playing).
9 - Play sound continuously (until the PlaySound() function is called again).

Example(s)

This script plays the sound of the file c:\welcome.wav one time and holds script execution until it has finished playing.

```
PlaySound("c:\welcome.wav",0);
```

This script plays the sound Alert continuously. In the win.ini file [Sounds] section you need to associate the sound name Alert with a sound file, such as:

```
Alert=c:\alert.wav.
PlaySound("Alert",9);
```

PostLogonDialog() Function

Shows the InTouch **Logon** dialog box and returns TRUE.

Category

security

Syntax

```
DiscreteTag=PostLogonDialog();
```

Examples

```
DiscreteTag=PostLogonDialog();
```

See Also

InvisibleVerifyCredentials(), AttemptInvisibleLogon(), IsAssignedRole(), QueryGroupMembership(), AddPermission()

PrintScreen() Function

You can write a script to print the entire WindowViewer screen with the PrintScreen() function.

Syntax

```
PrintScreen (ScreenOption, PrintOption)
```

Parameters

ScreenOption
Determines how much of the WindowViewer screen is to be printed. A literal integer value, integer tagname, or integer expression.

1 - Print the client area, no menus (default)

2 - Print the entire window area, including menus

PrintOption
Determines how the printed image is to be stretched to fit on the printout.

- 1 - Best Fit:
  image is stretched so that it fits either horizontally or vertically on the printout without changing the aspect ratio. (default)
- 2 - Vertical Fit:
  image is stretched so that it fits vertically on the printout without changing the aspect ratio. The image may be cut off horizontally.
- 3 - Horizontal Fit:
  image is stretched so that it fits horizontally on the printout without changing the aspect ratio. The image may be cut off vertically.
- 4 - Stretch to Page:
  image is stretched so that it fits horizontally and vertically on the printout. The aspect ratio may change but the image is not truncated.
- Invalid options, including 0, default to Best Fit.

Note: Popup windows that extend beyond the WindowViewer screen area are cut off.

Example(s)

This script sends a printout of the current entire WindowViewer screen area without menus to the printer queue. The printout contains the screen area stretched so that it fills the printout dimensions.

```
PrintScreen(1,4);
```

QueryGroupMembership() Function

Determines whether the currently logged on user is a member of the specified user group. Only applies to operating system security.

Category

security

Syntax

```
DiscreteTag=QueryGroupMembership( "Domain", "Group" );
```

Arguments

Domain
Name of the domain or local computer in which the group is located

Group
Name of the group.

Remarks

Valid for operating system security mode only and applies to the currently logged on user. If a user is currently logged on and if he or she is part of the group located on the domain, then TRUE is returned. Otherwise, FALSE is returned.

The QueryGroupMembership() function works with operating system-based security and with ArchestrA security only when the ArchestrA security is set to operating system-based security.

Examples

```
DiscreteTag=QueryGroupMembership( "corporate_hq", "InTouchAdmins" );
DiscreteTag=QueryGroupMembership( "JohnS01", "InTouchUsers" );
```

See Also

PostLogonDialog(), InvisibleVerifyCredentials(), IsAssignedRole(), AttemptInvisibleLogon(), AddPermission()

ShowHome() Function

Opens the InTouch window(s) you specified in the **Home Windows** tab in the WindowViewer **Properties** dialog box and closes any other windows.

Syntax

```
ShowHome();
```

Starting a Windows Application

In a script, you can start a Windows application using the StartApp command.

Syntax

```
StartApp appname;
```

Parameters

appname
Path and file name of the application you want to start. A literal string value, message tagname, or string expression.

Note: You need to know the path and file name of the application. If the application is in a directory that is part of the Windows PATH environment variable, you only need to pass the file name (without path).

Example(s)

This script starts Microsoft Calculator.

```
StartApp "calc"
```

SwitchDisplayLanguage() Function

Switches the display of visible, static texts and alarm fields in a desired language for which translated strings are provided.

Category

misc

Syntax

```
SwitchDisplayLanguage(LocaleID);
```

Parameter

LocaleID
The language in which static text strings and alarm fields are to be shown at run time.

Example(s)

In this example, German is the language to be shown at run time.

```
SwitchDisplayLanguage(1031);
```

See Also

$Language system tag

TseGetClientId() Function

Returns a string version of the client ID (the TCP/IP address of the client) if the View application is running on a Terminal Server client. This client ID is used internally to generate SuiteLink server names and logger file names. Otherwise, the TseGetClientId() function returns an empty string.

Syntax

```
MessageResult=TseGetClientId();
```

Example

The client IP address 10.103.202.1 is saved to the MsgTag tag.

```
MsgTag=TseGetClientID();
```

TseGetClientNodeName() Function

Returns the client node name if the View application is running on a Terminal Server client assigned a name that can be identified by Windows. Otherwise, the TseGetClientNodeName() function returns an empty string.

Syntax

```
MessageResult=TseGetClientNodeName();
```

Example

The client node name is returned as the value assigned to the MsgTag tag.

```
MsgTag=TseGetClientNodeName();
```

TseQueryRunningOnClient() Function

Returns a non-zero integer value if the View application is running on a Terminal Services client. Otherwise, it returns a zero.

Syntax

```
Result=TseQueryRunningOnClient();
```

Return Value

Returns 0 if View is not running on a Terminal Services client.

Example

IntTag is set to 1 if WindowViewer is running on a Terminal Services client.

```
IntTag=TseQueryRunningOnClient;
```

TseQueryRunningOnConsole() Function

The TseQueryRunningOnConsole() function can be run from a script to indicate whether the View application is running on a Terminal Services console.

Syntax

```
Result=TseQueryRunningOnConsole();
```

Return Value

Returns a non-zero integer value if the View application is running on a Terminal Services console. Otherwise, the TseQueryRunningOnConsole() function returns a zero.

Example

IntTag is set to 1 if WindowViewer is running on a Terminal Services console.

```
IntTag=TseQueryRunningOnConsole();
```

Math Functions

Use math functions to return the answer to the specified mathematical expression.

In QuickScript, all mathematical operations are calculated internally as double, regardless of the operand data type. Following standard mathematical rules, the result is always rounded in division operations to maintain accuracy. Rounding only occurs on the end result, not intermediate values, and the quotient will match the target data type. This is the standard methodology for SCADA and DCS systems, and provides the data integrity, precision retention, time stamps, and overall data quality propagation and aggregation needed for these systems.

If you want to round at each step instead of only at the final result, you can leverage the support built into QuickScript for .NET libraries and utilize the System.Math.Floor and System.Math.Round methods to explicitly round the intermediate steps. As an example, consider the following script:

```
dim dividend as integer;
dim divisor as integer;
dim quotient as integer;
dim remainder as integer;
dividend = 8;
divisor = 3;
LogMessage("Value of dividend = " + dividend);
LogMessage("Value of divisor = " + divisor);
quotient = dividend/divisor;
LogMessage("Value of quotient = " + quotient);
remainder = dividend mod divisor;
LogMessage ("Value of remainder = " + remainder);
dividend = divisor*quotient +remainder;
LogMessage ("Value of dividend = " + dividend);
```

The result is: 8 / 3 = 3

If, instead, you want to drop the remainder (not rounding the final result to the nearest integer), you could add a call to the Math.Floor method and use the following:

```
dim dividend as integer;
dim divisor as integer;
dim quotient as integer;
dim remainder as integer;
dividend = 8;
divisor = 3;
LogMessage("Value of dividend = " + dividend);
LogMessage("Value of divisor = " + divisor);

// *** Add call to Math.Floor. This drops the remainder rather than rounding the
internal Double result to integer
quotient = System.Math.Floor(dividend/divisor);

LogMessage("Value of quotient = " + quotient);
remainder = dividend mod divisor;
LogMessage ("Value of remainder = " + remainder);
dividend = divisor*quotient +remainder;
LogMessage ("Value of dividend = " + dividend);
```

The result is: 8 / 3 = 2 (remainder 2)

Abs()

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

Category

Math

Syntax

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

Parameter

Number
Any number or numeric attribute.

Examples

```
Abs(14); ' returns 14
Abs(-7.5); ' returns 7.5
```

ArcCos()

Returns an angle between 0 and 180 degrees whose cosine is equal to the number specified.

Category

Math

Syntax

```
Result = ArcCos( Number );
```

Parameter

Number
Any number or numeric attribute with a value between -1 and 1 (inclusive).

Examples

```
ArcCos(1); ' returns 0
ArcCos(-1); ' returns 180
```

See Also

Cos() on page 60, *Sin()* on page 63, *Tan()* on page 64, *ArcSin()* on page 59, *ArcTan()* on page 60

ArcSin()

Returns an angle between -90 and 90 degrees whose sine is equal to the number specified.

Category

Math

Syntax

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

Parameter

Number
Any number or numeric attribute with a value between -1 and 1 (inclusive).

Examples

```
ArcSin(1); ' returns 90
ArcSin(-1); ' returns -90
```

See Also

Cos() on page 60, *Sin()* on page 63, *Tan()* on page 64, *ArcCos()* on page 59, *ArcTan()* on page 60

ArcTan()

Returns an angle between -90 and 90 degrees whose tangent is equal to the number specified.

Category

Math

Syntax

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

Parameter

Number
Any number or numeric attribute.

Examples

```
ArcTan(1); ' returns 45
ArcTan(0); ' returns 0
```

See Also

Cos() on page 60, *Sin()* on page 63, *Tan()* on page 64, *ArcCos()* on page 59, *ArcSin()* on page 59

Cos()

Returns the cosine of an angle in degrees.

Category

Math

Syntax

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

Parameter

Number
Any number or numeric attribute.

Examples

```
Cos(90); ' returns 0
Cos(0); ' returns 1
```

This example shows how to use the function in a math equation:

```
Wave = 50 * Cos(6 * Now().Second);
```

See Also

Sin() on page 63, *Tan()* on page 64, *ArcCos()* on page 59, *ArcSin()* on page 59, *ArcTan()* on page 60

Exp()

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

Category

Math

Syntax

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

Parameter

Number
Any number or numeric attribute.

Example

```
Exp(1); ' returns 2.718...
```

Int()

Returns the next integer less than or equal to a specified number.

Category

Math

Syntax

```
IntegerResult = Int( Number );
```

Parameter

Number
Any number or numeric attribute.

Remarks

When handling negative real (float) numbers, this function returns the integer farthest from zero.

Examples

```
Int(4.7); ' returns 4
Int(-4.7); ' returns -5
```

Log()

Returns the natural log (base e) of a number.

Category

Math

Syntax

```
RealResult = Log( Number );
```

Parameter

Number
Any number or numeric attribute.

Remarks

Natural log of 0 is undefined.

Examples

```
Log(100); ' returns 4.605...
Log(1); ' returns 0
```

See Also

LogN() on page 62, *Log10()* on page 61

Log10()

Returns the base 10 log of a number.

Category

Math

Syntax

```
Result = Log10( Number );
```

Parameter

Number
Any number or numeric attribute.

Example

```
Log10(100); ' returns 2
```

See Also

Log() on page 61, *LogN()* on page 62

LogN()

Returns the values of the logarithm of x to base n.

Category

Math

Syntax

```
Result = LogN( Number, Base );
```

Parameters

Number
Any number or numeric attribute.

Base
Integer to set log base. You could also specify an integer attribute.

Remarks

Base 1 is undefined.

Examples

```
LogN(8, 3); ' returns 1.89279
LogN(3, 7); ' returns 0.564
```

See Also

Log() on page 61, *Log10()* on page 61

Pi()

Returns the value of Pi.

Category

Math

Syntax

```
RealResult = Pi();
```

Example

```
Pi(); ' returns 3.1415926
```

Round()

Rounds a real number to a specified precision and returns the result.

Category

Math

Syntax

```
RealResult = Round( Number, Precision );
```

Parameters

Number
Any number or numeric attribute.

Precision
Sets the precision to which the number is rounded. This value can be any number or a numeric attribute.

Examples

```
Round(4.3, 1); ' returns 4
Round(4.3, .01); ' returns 4.30
Round(4.5, 1); ' returns 5
Round(-4.5, 1); ' returns -4
Round(106, 5); ' returns 105
Round(43.7, .5); ' returns 43.5
```

See Also

Trunc() on page 65

Sgn()

Determines the sign of a value (whether it is positive, zero, or negative) and returns the result.

Category

Math

Syntax

```
IntegerResult = Sgn( Number );
```

Parameter

Number
Any number or numeric attribute.

Return Value

If the input number is positive, the result is 1. Negative numbers return a -1, and 0 returns a 0.

Examples

```
Sgn(425); ' returns 1;
Sgn(0); ' returns 0;
Sgn(-37.3); ' returns -1;
```

Sin()

Returns the sine of an angle in degrees.

Category

Math

Syntax

```
Result = Sin( Number );
```

Parameter

Number
Angle in degrees. Any number or numeric attribute.

Examples

```
Sin(90); ' returns 1;
Sin(0); ' returns 0;
```

This example shows how to use the function in a math expression:

```
wave = 100 * Sin (6 * Now().Second);
```

See Also

Cos() on page 60, *Tan()* on page 64, *ArcCos()* on page 59, *ArcSin()* on page 59, *ArcTan()* on page 60

Sqrt()

Returns the square root of a number.

Category

Math

Syntax

```
RealResult = Sqrt( Number );
```

Parameter

Number
Any number or numeric attribute.

Example

This example takes the value of me.PV and returns the square root as the value of x:

```
x=Sqrt(me.PV);
```

Tan()

Returns the tangent of an angle given in degrees.

Category

Math

Syntax

```
Result = Tan( Number );
```

Parameter

Number
The angle in degrees. Any number or numeric attribute.

Examples

```
Tan(45); ' returns 1;
Tan(0); ' returns 0;
```

This example shows how to use the function in a math expression:

```
Wave = 10 + 50 * Tan(6 * Now().Second);
```

See Also

Cos() on page 60, *Sin()* on page 63, *ArcCos()* on page 59, *ArcSin()* on page 59, *ArcTan()* on page 60

Trunc()

Truncates a real (floating point) number by simply eliminating the portion to the right of the decimal point, including the decimal point, and returns the result.

Category

Math

Syntax

```
NumericResult = Trunc( Number );
```

Parameter

Number
Any number or numeric attribute.

Remarks

This function accomplishes the same result as placing the contents of a float type attribute into an integer type attribute.

Examples

```
Trunc(4.3); ' returns 4;
Trunc(-4.3); ' returns -4;
```

See Also

Round() on page 63

Miscellaneous Functions

Functions in the miscellaneous group perform a variety of purposes, such as logging data or querying attributes.

ActivateApp()

Restores, minimizes, maximizes, or closes another currently running Windows application.

Category

Miscellaneous

Syntax

```
ActivateApp( TaskName );
```

Parameter

TaskName
The task this function activates.

Remarks

TaskName is the exact text string, including spaces, that appears on the Task Bar or in Windows Task Manager. You can see the task name by opening Task Manager.

Example

```
ActivateApp("Calculator");
```

Filtering Events

To get only specific events, filters can be introduced before getting events from the event service. The filtering should be done before the StartRequestingEvent() method is called.

The following datatypes are supported when filtering the events.

- Integer
- Float
- String
- Bool
- DateTime
- Double
- Short
- Array

The following table shows the comparison types that are supported for filtering events.

Comparison Keyword	Description
eq	Means EqualTo. Returns all the events matching the filtered criteria.
beginswith	Means StartsWith. Returns all the events matching the filtered criteria. Applies only to string data type filtering
lt	Means Lesser Than. Applies to all supported data types excluding string. It does not support arrays.
le	Means Lesser or Equal. Applies to all supported data types excluding string. It does not support arrays.
gt	Means Greater Than. Applies to all supported data types excluding string. It does not support arrays.
ge	Means Greater or Equal. Applies to all supported data types excluding string. It does not support arrays.
between	Checks will be made only to paired supplied values. Returns all the events matching the filtered criteria. It supports numeric and date data types.
neg, nbegins, nlt, nle, ngt, nge, nbetween	A keyword 'n' before the comparison keyword Means NOT of.

DateTimeGMT()

Returns a number representing the number of days and fractions of days since January 1, 1970, in Coordinated Universal Time (UTC), regardless of the local time zone.

Category

Miscellaneous

Syntax

```
Result=DateTimeGMT();
```

Parameters

```
None
```

Example

```
MessageTag = StringFromTime(DateTimeGMT() * 86400.0, 3);
```

IsBad()

Returns a Boolean value indicating if the quality of the specified attribute is Bad.

Category

Miscellaneous

Syntax

```
BooleanResult = IsBad( Attribute1, Attribute2, … );
```

Parameter(s)

Attribute1, Attribute2, ...AttributeN
Names of one or more attributes for which you want to determine Bad quality. You can include a variable-length list of attributes.

Return Value

If any of the specified attributes has Bad quality, then true is returned. Otherwise, false is returned.

Examples

```
IsBad(TIC101.PV);
IsBad(TIC101.PV, PIC102.PV);
```

See Also

IsGood() on page 67, *IsInitializing()* on page 67, *IsUncertain()* on page 68, *IsUsable()* on page 68

IsGood()

Returns a Boolean value indicating if the quality of the specified attribute is Good.

Category

Miscellaneous

Syntax

```
BooleanResult = IsGood( Attribute1, Attribute2, … );
```

Parameter(s)

Attribute1, Attribute2, and so on
Name of the attribute(s) for which you want to determine Good quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have Good quality, then true is returned. Otherwise, false is returned.

Examples

```
IsGood(TIC101.PV);
IsGood(TIC101.PV, PIC102.PV);
```

See Also

IsBad() on page 67, *IsInitializing()* on page 67, *IsUncertain()* on page 68, *IsUsable()* on page 68

IsInitializing()

Returns a Boolean value indicating if the quality of the specified attribute is Initializing.

Category

Miscellaneous

Syntax

```
BooleanResult = IsInitializing( Attribute1, Attribute2, … );
```

Parameter(s)

Attribute1, Attribute2, and so on
Name of the attribute(s) for which to determine Initializing quality. You can include a variable-length list of attributes.

Return Value

If any of the specified attributes has Initializing quality, then true is returned. Otherwise, false is returned.

Examples

```
IsInitializing(TIC101.PV);
IsInitializing(TIC101.PV, PIC102.PV);
```

See Also

IsBad() on page 67, *IsGood()* on page 67, *IsUncertain()* on page 68, *IsUsable()* on page 68

IsUncertain()

Returns a Boolean value indicating if the quality of the specified attribute is Uncertain.

Category

Miscellaneous

Syntax

```
BooleanResult = IsUncertain( Attribute1, Attribute2, … );
```

Parameter(s)

Attribute1, Attribute2, and so on
Name of the attribute(s) to determine Uncertain quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have Uncertain quality, then true is returned. Otherwise, false is returned.

Examples

```
IsUncertain(TIC101.PV);
IsUncertain(TIC101.PV, PIC102.PV);
```

See Also

IsBad() on page 67, *IsGood()* on page 67, *IsInitializing()* on page 67, *IsUsable()* on page 68

IsUsable()

Returns a Boolean value indicating if the specified attribute is usable for calculations.

Category

Miscellaneous

Syntax

```
BooleanResult = IsUsable( Attribute1, Attribute2, … );
```

Parameter(s)

Attribute1, Attribute2, ...AttributeN
Name of one or more attributes for which you want to determine unusable quality. You can include a variable-length list of attributes.

Return Value

If all of the specified attributes have either Good or Uncertain quality, then true is returned. Otherwise, false is returned.

Remarks

To qualify as usable, the attribute must have Good or Uncertain quality. In addition, each float or double attribute cannot be a NaN (not a number).

Examples

```
IsUsable(TIC101.PV);
IsUsable(TIC101.PV, PIC102.PV);
```

See Also

IsBad() on page 67, *IsGood()* on page 67, *IsInitializing()* on page 67, *IsUncertain()* on page 68

LogCustom()

Writes a user-defined custom flag message in the Log Viewer.

Category

Miscellaneous

Syntax

```
LogCustom( CustomFlag, msg );
```

Parameter

CustomFlag
Creates a new log flag based on the first parameter string. The first call creates the custom flag.

msg
The message to write to the Log Viewer. Actual string or a string attribute.

Remarks

The log flag is disabled by default.

The message is always logged under the component "ObjectName.ScriptName". For example, "WinPlatform_001.script1: msg", which identifies what object and what script within the object logged the error.

LogCustom() is similar to LogMessage(), but displays the message in the custom log flag when Log Custom is enabled.

The parameter help tooltip and Function Browser sample parameter list will show "LogCustom(CustomFlag, msg)" rather than "LogCustom(CustomFlag, Message)". "Message" is a reserved keyword.

Example

```
LogCustom(EditBox1.text, "User-defined message.";
```

This statement writes to the Log Viewer as follows:

```
10/24/2005 12:49:14 PM ScriptRuntime
<ObjectName.ScriptName>: <LogFlag EditBox1> User-defined message.
```

LogDataChangeEvent()

Logs an application change event to the Galaxy Historian.

Note: The LogDataChangeEvent() function works only in object scripts, not in symbol scripts.

Category

Miscellaneous

Syntax

```
LogDataChangeEvent(AttributeName, Description, OldValue, NewValue, TimeStamp);
```

Parameters

AttributeName
Attribute name as a tag name.

Description
Description of the object.

OldValue
Old value of the attribute.

NewValue
New value of the attribute.

TimeStamp
The time stamp associated with the logged event. The timestamp can be UTC or local time. The TimeStamp parameter is optional. The timestamp of the logged event defaults to Now() if a TimeStamp parameter is not included.

Remarks

A symbol script still compiles if the LogDataChangeEvent() function is included. However, a warning message is written to the log at run time that the function is inoperable.

Example

This example logs an event when a pump starts or stops with a timestamp of the current time when the event occurred.

```
LogDataChangeEvent(TC104.pumpstate, "Pump04", OldState, NewState);
```

LogError()

Writes a user-defined error message in the Log Viewer with a red error log flag.

Category

Miscellaneous

Syntax

```
LogError( msg );
```

Parameter

msg
The message to write to the Log Viewer. Actual string or a string attribute.

Remarks

The log flag is enabled by default.

The message is always logged under the component "ObjectName.ScriptName". For example, "WinPlatform_001.script1: msg", which identifies what object and what script within the object logged the error.

LogError() is similar to LogMessage(), but displays the message in red.

The parameter help tooltip and Function Browser sample parameter list will show "LogError(msg)" rather than "LogError(Message)". "Message" is a reserved keyword.

Example

```
LogError("User-defined error message.");
```

This statement writes to the Log Viewer as follows:

```
10/24/2005 12:49:14 PM ScriptRuntime
<ObjectName.ScriptName>: User-defined error message.
```

LogMessage()

Writes a user-defined message to the Log Viewer.

Category

Miscellaneous

Syntax

```
LogMessage( msg );
```

Parameter

msg
The message to write to the Log Viewer. Actual string or a string attribute.

Remarks

This is a very powerful function for troubleshooting 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 attributes both before they are changed and after they are affected by the script.

Each message posted to the Log Viewer is stamped with the exact date and time. The message always begins with the component "Tagname.ScriptName" so you can tell what object and what script within the object posted the message to the log.

Examples

```
LogMessage("Report Script is Running");
```

The above statement writes the following to the Log Viewer:

```
10/24/2005 12:49:14 PM ScriptRuntime <Tagname.ScriptName>:Report Script is
Running.
MyTag=MyTag + 10;
LogMessage("The Value of MyTag is " + Text(MyTag, "#"));
```

LogTrace()

Writes a user-defined trace message in the Log Viewer.

Category

Miscellaneous

Syntax

```
LogTrace( msg );
```

Parameter

msg
The message to write to the Log Viewer. Actual string or a string attribute.

Remarks

The log flag is disabled by default.

The message is always logged under the component "ObjectName.ScriptName". For example, "WinPlatform_001.script1: msg", which identifies what object and what script within the object logged the error.

LogTrace() is similar to LogMessage(), but displays the message as Trace when Log Trace is enabled.

The parameter help tooltip and Function Browser sample parameter list will show "LogTrace(msg)" rather than "LogTrace(Message)". "Message" is a reserved keyword.

Example

```
LogTrace("User-defined trace message.");
```

This statement writes to the Log Viewer as follows:

```
10/24/2005 12:49:14 PM ScriptRuntime
<ObjectName.ScriptName>: User-defined trace message.
```

LogWarning()

Writes a user-defined error message in the Log Viewer with a yellow warning log flag.

Category

Miscellaneous

Syntax

```
LogWarning( msg );
```

Parameter

msg
The message to write to the Log Viewer. Actual string or a string attribute.

Remarks

The log flag is disabled by default.

The message is always logged under the component "ObjectName.ScriptName". For example, "WinPlatform_001.script1: msg", which identifies what object and what script within the object logged the error.

LogWarning() is similar to LogMessage(), but displays the message as a yellow warning message.

The parameter help tooltip and Function Browser sample parameter list will show "LogWarning(msg)" rather than "LogWarning(Message)". "Message" is a reserved keyword.

Example

```
LogWarning("User-defined warning message.")
```

This statement writes to the Log Viewer as follows:

```
10/24/2005 12:49:14 PM ScriptRuntime
<ObjectName.ScriptName>: User-defined warning message.
```

SendKeys()

Sends keystrokes to an application. To the receiving application, the keys appear to be entered from the keyboard. You can use SendKeys() within a script to enter data or send commands to an 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.

Category

Miscellaneous

Syntax

```
SendKeys( KeySequence );
```

Parameter

KeySequence
Any key sequence or a string attribute.

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 key codes for unique keyboard keys:

Key	Code
BACKSPACE	{BACKSPACE}or {BS}
BREAK	{BREAK}
CAPSLOCK	{CAPSLOCK}
DELETE	{DELETE} or {DEL}
DOWN	{DOWN}
END	{END}
ENTER	{ENTER} or tilde (~)
ESCAPE	{ESCAPE} or {ESC}
F1...F12	{F1}...{F12}
HOME	{HOME}
INSERT	{INSERT}
LEFT	{LEFT}
NUMLOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRTSC	{PRTSC}
RIGHT	{RIGHT}
TAB	{TAB}
UP	{UP}
HOME	{HOME}

Special keys (SHIFT, CTRL, and ALT) have their own key codes:

Key	Code
SHIFT	+ (plus)
CTRL	^ (caret)
ALT	% (percent)

Enhancements to the Microsoft Hardware Abstraction Layer in Windows prevents the SendKeys() function from operating on some computers.

Examples

To use two special keys together, use a second set of parentheses. The following statement holds 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 Calculator and sends the key combination 1234:

```
ActivateApp("Calculator");
SendKeys("^(1234)");
```

SetAttributeVT()

Sets the value and timestamp of an object attribute. For buffered values, only the last calculated value is captured for historization.

Category

Miscellaneous

Syntax

```
SetAttributeVT( Attribute, Value, TimeStamp);
```

Parameter

Attribute
Name of the object attribute whose value and timestamp are modified. The specified attribute must belong to the object to which the script is attached.

Value
Value of the attribute, which can be a reference. The quality is always set to Good.

TimeStamp
Timestamp that can be a reference, a variable, or a string interpreted as the computer's local time or UTC. The timestamp is converted internally to UTC format before the attribute's value is sent to the run-time component.

Remarks

Interim calculated buffered values are NOT historized. Use SetAttributeVT2() if historization of interim values is needed.

Timestamp can be set only for object attributes that support a timestamp. At compile time, the script cannot detect whether the attribute specified with the SetAttributeVT() function supports a timestamp or not. No warning is issued if the attribute does not support a timestamp.

Example

This example sets an integer value and timestamp for an attribute that indicates pump RPM.

```
SetAttributeVT(me.PV, TC104.PumpRPM, LCLTIME);
```

SetAttributeVT2()

Sets the value and timestamp of an object attribute. This function is identical to SetAttributeVT(), but SetAttributeVT2() allows interim calculated data for buffered values to be historized once per scan cycle.

Category

Miscellaneous

Syntax

```
SetAttributeVT2( Attribute, Value, TimeStamp);
```

Parameter

Attribute
Name of the object attribute whose value and timestamp are modified. The specified attribute must belong to the object to which the script is attached.

Value
Value of the attribute, which can be a reference. The quality is always set to Good.

TimeStamp
Timestamp that can be a reference, a variable, or a string interpreted as the computer's local time or UTC. The timestamp is converted internally to UTC format before the attribute's value is sent to the run-time component.

Remarks

In contrast to SetAttributeVT(), SetAttributeVT2() allows historization of interim calculated buffered values.

Timestamp can be set only for object attributes that support a timestamp. At compile time, the script cannot detect whether the attribute specified with the SetAttributeVT2() function supports a timestamp or not. No warning is issued if the attribute does not support a timestamp.

Example

This example sets an integer value and timestamp for an attribute that indicates pump RPM (interim calculated values for buffered data are historized).

```
SetAttributeVT2(me.PV, TC104.PumpRPM, LCLTIME);
```

SetBad()

Sets the quality of an attribute to Bad.

Category

Miscellaneous

Syntax

```
SetBad( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Bad.

Remarks

The specified attribute must be within the object to which the script is attached.

Example

```
SetBad(me.PV);
```

See Also

SetGood() on page 75, *SetInitializing()* on page 76, *SetUncertain()* on page 76

SetGood()

Sets the quality of an attribute to Good.

Category

Miscellaneous

Syntax

```
SetGood( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Good.

Remarks

The specified attribute must be within the object to which the script is attached.

Example

```
SetGood(me.PV);
```

See Also

SetBad() on page 75, *SetInitializing()* on page 76, *SetUncertain()* on page 76

SetInitializing()

Sets the quality of an attribute to Initializing.

Category

Miscellaneous

Syntax

```
SetInitializing( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Initializing.

Remarks

The specified attribute must be within the object to which the script is attached.

Example

```
SetInitializing(me.PV);
```

See Also

SetBad() on page 75, *SetGood()* on page 75, *SetUncertain()* on page 76

SetUncertain()

Sets the quality of an attribute to Uncertain.

Category

Miscellaneous

Syntax

```
SetUncertain( Attribute );
```

Parameter

Attribute
The attribute for which you want to set the quality to Uncertain.

Remarks

The specified attribute must be within the object to which the script is attached.

Example

```
SetUncertain(me.PV);
```

See Also

SetBad() on page 75, *SetGood()* on page 75, *SetInitializing()* on page 76

SignedAlarmAck()

Acknowledges one or more alarms on ArchestrA attributes, optionally requiring a signature if any of the indicated alarms falls within a designated priority range.

This function is supported only for client scripting and not object scripting.

Category

Miscellaneous

Syntax

```
int SignedAlarmAck(String Alarm_List,
Boolean Signature_Reqd_for_Range,
Integer Min_Priority,
Integer Max_Priority,
String Default_Ack_Comment,
Boolean Ack_Comment_Is_Editable,
String TitleBar_Caption,
String Message_Caption
);
```

Parameters

Alarm_List

The list of alarms to be acknowledged. The list must be a single text string with each alarm name separated by a space or a comma.

Data Type
String

Valid Range
Limit 1024 characters

Additional Information
Can be a constant string, a reference, or an expression.
Only alarms on ArchestrA attributes are supported.
If there is any invalid alarm in the list, then none of the alarms are acknowledged.

Examples
Example 1:

```
"UD1.analog_001.HiHi"
```

The collection is represented as a text string, with alarms separated by blanks and/or commas.

Example 2:

```
"UD1.analog_001.HiHi UD9.x14.dev.major"
```

Example 3:

```
"UD1.analog_001.HiHi, UD9.x14.dev.major"
```

Example 4, an array of strings such as:

```
Pump1.AlarmArray[1] = "Pump1.Level.HiHi"
```

```
Pump1.AlarmArray[2] = "Pump1.Level.LoLo"
```

uses the function as follows:

```
SignedAlarmAck(Pump1.AlarmArray[ ], ...)
```

The script passes to the function the following single string:

```
"Pump1.Level.HiHi, Pump1.Level.LoLo"
```

Signature_Reqd_for_Range

Indicates whether a signature is required for acknowledging alarms.

Data Type
Bool

Additional Information
Can be a constant, a reference, or an expression.

Min_Priority

Represents the minimum priority value of the range for which the signature is required.

Data Type
Integer

Valid Range
1-999; must be less than or equal to the Max_Priority value.

Additional Information
Can be a constant, a reference, or an expression.

Max_Priority

Represents the maximum priority value of the range for which the signature is required.

Data Type
Integer

Valid Range
1-999; must be greater than or equal to the Min_Priority value.

Additional Information
Can be a constant, a reference, or an expression.

Default_Ack_Comment

Comment to be shown in the **Acknowledge Alarms** dialog box.

Data Type
String

Valid Range
Limit 200 characters

Additional Information
Can be a constant, a reference or an expression.
If the parameter is empty, then no default comment is shown in the **Acknowledge Alarms** dialog box.

Ack_Comment_Is_ Editable

Indicates whether the run-time user can modify the acknowledgement comment.

Data Type
Bool

Additional Information
Can be a constant, a reference, or an expression.
If set to False, the **Comment** box in the **Acknowledge Alarms** dialog box is unavailable.

TitleBar_Caption

Shows a title in the title bar of the **Acknowledge Alarms** dialog box.

Data Type
String

Valid Range
Limi 1024 characters

Additional Information
Can be a constant, a reference, or an expression.

If the TitleBar_Caption is empty, the default title, Acknowledge Alarms**, is shown.**

Message_Caption

Shows a customizable message to the run-time user in the **Acknowledge Alarms** dialog box.

Data Type
String

Valid Range
Limit 250 characters

Additional Information
Can be a constant, a reference, or an expression.
Use the parameter to provide more information on the alarm to the run-time user.
This message is not propagated to the event record.

Return Values

Return values indicate success or failure status. A non-zero value indicates type of failure.

-1 The user canceled the operation.

The function writes a message to the Logger indicating user cancellation.

-2 No alarms are waiting for acknowledgement.

0 The function is successful and the following are all true:

- The function parameters are valid.
- The user credentials are valid (or no credentials are needed).
- The user did not cancel the operation.
- Function wrote to the .AckMsg attributes of the indicated alarms.

1 The function failed due to any error that is not covered by the other specified return values.

2 One or more parameters were not coerced to the appropriate data type at run time.

Example: Parameter is a reference with Boolean as the expected data type. At run time, reference is to a String data type that cannot be coerced to True or False.

The function writes a message to the Logger.

3 The Alarm_List parameter was not valid at run time.

- String was null, empty or contained no attribute references.
- Contained one or more items that were not valid attribute references.
- Contained one or more attribute references that did not exist or did not identify valid alarm primitives.

If Alarm_List contains a mixture of valid and invalid references, the function does nothing. The function does not attempt to operate on the valid references, and returns this error status.

4 The Min_Priority or MaxPriority values do not fall within the range of 1 to 999.

The function writes a message to the Logger indicating which parameter was out of range and showing the actual value.

5 The Min_Priority value is greater than the Max_Priority value.

The function writes a message to the Logger identifying the problem and showing the actual values.

Note: A return value of zero does not indicate if the alarms are acknowledged, only that the function wrote to the AckMsg attributes. The alarms may not be acknowledged due to insufficient permission or if the alarms have already been acknowledged.

Remarks

For more information about using the SignedAlarmAck() function, see the topic *Signature Security for Acknowledging Alarms,* under "Adding and Maintaining Symbol Scripts" in the *Creating and Managing Industrial Graphics User Guide*.

Examples

```
Dim n as Integer;
n = SignedAlarmAck("UD1.analog_001.HiHi UD9.x14.dev.major", true, 1, 250,
"Acknowledged by script", true, "Acking Tank Alarms", "Acknowledge the tank
alarms");
```

Using an array of strings:

```
dim arr[2] as String;
arr[1] = "UD1.analog_001.HiHi";
arr[2] = "UD9.x14.dev.major";
n = SignedAlarmAck(arr[], true, 200, 500, "Acked by script", true, "Acking
Tank Alarms", "Please acknowledge the tank alarms.");
```

SignedWrite()

Performs a write to an AutomationObject attribute that has a Secured Write or Verified Write security classification.

Category

Miscellaneous

Syntax

```
int SignedWrite(string Attribute,
object Value,
string ReasonDescription,
Bool Comment_Is_Editable,
```

```
Enum Comment_Enforcement,
string[] Predefined_Comment_List
);
```

Brackets [ ] indicate an array.

Parameters

Attribute

The attribute to be updated.

Data Type
String

Additional Information
Can be a constant string, a reference, or an expression.
Supports bound and nested bound references.
For detailed examples of Attribute parameter uses, see the topic *Examples of Using the Attribute Parameter in the SignedWrite() Function* under "Managing Symbols" in the *Creating and Managing Industrial Graphics User Guide.*

Examples
Example 1:

```
"UserDefined_001.temp"
```

Example 2:

```
"Pump15" + ".valve4"
```

Example 3:

With UDO_7 containing two string attributes, namestrA and namestrB set to the values "Tank1" and "Tank5" respectively, the following script writes to Tank1.Level or Tank5.Level according to whether strselect is "A" or "B":

```
Dim strselect As String;
Dim x As Indirect;
{ logic to set strselect to "A" or "B" }
x.BindTo ("UDO_7.namestr" + strselect);
SignedWrite(x + ".Level", 243, "Set " + x + " Level", true, 0, null);
```

Value

The value to be written.

Data Type
Object

Valid Range
Must match data type of the attribute being updated.

Additional Information
Can be a constant value, a reference, an expression, or NULL if nothing is to be entered.

ReasonDescription

Text that explains the purpose of the target attribute and the impact of changing it.

Data Type
String

Valid Range
Maximum of 256 characters.

Additional Information
Can be a constant string, a reference, or an expression.

The ReasonDescription is passed to the indicated Attribute as part of the write operation. The object also includes the user's write comment, if any. A Field Attribute description is used for the ReasonDescription parameter only if the attribute is a Field Attribute and it has a description (is not null). Otherwise, the Short Description for the corresponding ApplicationObject is used for the ReasonDescription parameter.

Comment_is _Editable

Indicates whether user can edit the write comment.

Data Type
Bool

Additional Information
Can be a constant value, a reference, or an expression.
If set to True: The comment text box is enabled with exceptions. If Comment_Is_Editable is true and if the Comment_ Enforcement parameter is PredefinedOnly, the comment text box is disabled. At run time, the user can only select a comment from the predefined comment list.
If the Comment_ Enforcement parameter is not PredefinedOnly, the comment list and box are enabled. You can select a comment from the comment list and modify it in the comment box.
If the predefined list is empty, the comment list is not shown in the dialog box.
If set to False: The predefined comment list does not appear in the Secured Write or Verified Write dialog boxes. The editable comment text box is disabled.

Comment_Enforcement

Contains choices of Optional, Mandatory and PredefinedOnly.

Data Type
Enum

Enumerations
Optional = 0
The run-time user can enter a comment or leave it blank.
Mandatory = 1
The run-time user must add a comment, either by selecting from the comment list or by entering a comment in the comment box.
PredefinedOnly = 2
The run-time user can select a comment from the comment list only. The comment text box is disabled.

Additional Information
Can be a constant, a reference, or an expression.

Predefined_Comment_List

An array of strings that can be used as predefined comments.

Data Type
String[]

Valid Range
Maximum of 20 comments, each with a maximum of 200 characters.

Additional Information
The array can be empty (number of elements is 0).
Can be a constant, a reference, an expression, or NULL if empty. Can reference an attribute that contains an array of strings.
If no predefined comment is entered, the predefined comment list is disabled at run time.
If Comment_Is_Editable is False, the predefined comment is still placed in the editable comment text box, but the user cannot modify it at run time.

Return Values

Return values indicate success or failure status. A non-zero value indicates type of failure.

0 The function returns a value of 0 (meaning success) if the following are all true:

- The function parameters were valid.
- The write operation was successfully placed on the queue for Secured and Verified Writes.
- If the user cancels the operation, a message is written to the Logger indicating user cancellation.

1 The function failed due to any error that is not covered by the other specified return values. This includes any error that is not covered by the other specified return values. If there is a failure, a specific message is logged in the Logger.

2 One or more parameters were not coerced to the appropriate data type at run time.

Example: Parameter is a reference with Boolean as the expected data type. At run time, reference is to a String data type that cannot be coerced to True or False. The function returns this value and writes a message to the Logger.

3 The attribute parameter was not valid at run time.

- Attribute string was null, empty, or contained no attribute reference.
- Attribute string contained an item that was not a valid attribute reference.
- Attribute string contained an attribute reference that did not exist.
- Attribute string contained an attribute reference that was not of the Secured Write or Verified Write security classification.

The function writes a message to the Logger identifying the error and the invalid attribute string.

4 The Comment_Enforcement parameter value was out of the range of valid enumerators.

Remarks

The SignedWrite() function is supported only for client scripting and not for object scripting.

A return value of 0 does not indicate whether the attribute was updated, only that the function placed an entry on the queue to write to the attribute. The operator may decide to cancel the operation after the Secured Write or Verified write dialog box is presented. In this case the attribute is not updated and a message is placed in the Logger indicating that the user canceled the operation. Even if the user enters valid credentials and clicks **OK**, the attribute still might not have been updated because of inadequate permission or data coercion problems.

The SignedWrite() function supports the custom property passed as the first parameter with opened and closed quotation marks, "".

If you configure the custom property CP as shown in the following script, the function attempts to resolve CP and determine if it has a reference. If it has a reference, then the reference is retrieved and the write is performed on the reference.

```
SignedWrite("CP", value, reason, editable, enforcement, null);
```

For more information about using the SignedWrite() function, see the topic *Working with the SignedWrite() Function for Secured and Verified Writes* under "Managing Symbols" in the *Creating and Managing Industrial Graphics User Guide.*

Examples

```
SignedWrite ("UserDefined_001.temp", 185, "This will change the oven
temperature", true, 1, null);
```

The following example shows the user an array of predefined comments:

```
Dim n as Integer;
n = SignedWrite("UserDefined_001.temp", 185, "This will change the oven
temperature", true, 1, UserDefined_001.OvenCommentArray[ ]);
```

where UserDefined_001.OvenCommentArray is an attribute containing an array of strings.

WriteStatus()

Returns the enumerated write status of the last write to the specified attribute.

Category

Miscellaneous

Syntax

```
Result = WriteStatus( Attribute );
```

Parameter

Attribute
The attribute for which you want to return write status.

Return Value

The return statuses are:

- MxStatusOk
- MxStatusPending
- MxStatusWarning
- MxStatusCommunicationError
- MxStatusConfigurationError
- MxStatusOperationalError
- MxStatusSecurityError
- MxStatusSoftwareError
- MxStatusOtherError

Remarks

If the attribute has never been written to, this function returns MxStatusOk. This function always returns MxStatusOk for attributes that do not support a calculated (non-Good) quality.

Example

```
WriteStatus(TIC101.SP);
```

WWControl()

Restores, minimizes, maximizes, or closes an application.

Category

Miscellaneous

Syntax

```
WWControl( AppTitle, ControlType );
```

Parameters

AppTitle
The name of the application title to be controlled. Actual string or a string attribute.

ControlType
Determines how the application is controlled. Possible values are shown below. These actions are identical to clicking on their corresponding selections in the application's Control Menu. Actual string or a string attribute.

"Restore" = Activates and shows the application's window.

"Minimize" = Activates a window and shows it as an icon.

"Maximize" = Activates and shows the application's window.

"Close" = Closes an application.

Example

```
WWControl("Calculator","Restore");
```

See Also

ActivateApp() on page 65

String Functions

Use string functions to work with character strings and string values.

DText()

Returns one of two possible strings, depending on the value of the *Discrete* parameter.

Category

String

Syntax

```
StringResult = DText( Discrete, OnMsg, OffMsg );
```

Parameters

Discrete
A Boolean value or Boolean attribute.

OnMsg
The message that is shown when the value of *Discrete* equals true.

OffMsg
The message shown when *Discrete* equals false.

Example

```
StringResult = DText(me.temp > 150, "Too hot", "Just right");
```

StringASCII()

Returns the ASCII value of the first character in a specified string.

Category

String

Syntax

```
IntegerResult = StringASCII( Char );
```

Parameter

Char
Alphanumeric character or string or string attribute.

Remarks

When this function is processed, only the single character is tested or affected. If the string provided to StringASCII contains more than one character, only the first character of the string is tested.

Examples

```
StringASCII("A"); ' returns 65;
StringASCII("A Mixer is Running"); ' returns 65;
StringASCII("a mixer is running"); ' returns 97;
```

See Also

StringChar() on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringChar()

Returns the character corresponding to a specified ASCII code.

Category

String

Syntax

```
StringResult = StringChar( ASCII );
```

Parameter

ASCII
ASCII code or an integer attribute.

Remarks

Use the StringChar function to add ASCII characters not normally represented on the keyboard to a string attribute.

This function is also useful for SQL commands. The where expression sometimes requires double quotation marks around string values, so use StringChar(34).

Example

In this example, a [Carriage Return (13)] and [Line Feed (10)] are added to the end of StringAttribute 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.

```
ControlString = StringAttribute+StringChar(13)+StringChar(10);
```

StringCompare()

Compares a string value with another string.

Category

String

Syntax

```
StringCompare( Text1, Text2 );
```

Parameters

Text1
First string in the comparison.

Text2
Second string in the comparison.

Return Value

The return value is zero if the strings are identical, -1 if Text1's value is less than Text2, or 1 if Text1's value is greater than Text2.

Example

```
Result = StringCompare ("Text1","Text2"); (or)
Result = StringCompare (MText1,MText2);
Where Result is an Integer or Real tag and MText1 and MText2 are Memory Message
tags.
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringFromTimeLocal()* on page 91, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringCompareNoCase()

Compares a string value with another string and ignores the case.

Category

String

Syntax

```
SStringCompareNoCase( Text1, Text2 );
```

Parameters

Text1
First string in the comparison.

Text2
Second string in the comparison.

Return Value

The return value is zero if the strings are identical (ignoring case), -1 if Text1's value is less than Text2 (ignoring case), or 1 if Text1's value is greater than Text2 (ignoring case).

Example

```
Result = StringCompareNoCase ("Text1","TEXT1"); (or)
```

```
Result = StringCompareNoCase (MText1,MText2);
Where Result is an Integer or Real tag and MText1 and MText2 are Memory Message tags.
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringFromTimeLocal()* on page 91, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringFromGMTTimeToLocal()

Converts a time value (in seconds since Jan-01-1970) to a particular string representation. This is the same as StringFromTime().

Category

String

Syntax

```
MessageResult=StringFromGMTTimeToLocal(SecsSince1-1-70,StringType);
```

Parameters

SecsSince1-1-70
Is converted to the StringType specified and the result is stored in MessageResult.

StringType
Determines the display method:

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"

Remarks

Any adjustments necessary due to Daylight Savings Time are automatically applied to the return result. Therefore, it is not necessary to make any manual adjustments to the input value to convert to DST.

Example

This example assumes that the time zone on the local node is Pacific Standard Time (UTC-0800). The UTC time passed to the function is 12:00:00 AM on Friday, 1/2/1970. Since PST is 8 hours behind UTC, the function will return the following results:

```
StringFromGMTTimeToLocal(86400, 1); ' returns "1/1/1970"
StringFromGMTTimeToLocal(86400, 2); ' returns "04:00:00 PM"
StringFromGMTTimeToLocal(86400, 3); ' returns "Thu Jan 01 16:00:00 1970"
StringFromGMTTimeToLocal(86400, 4); ' returns "Thu"
StringFromGMTTimeToLocal(86400, 5); ' returns "Thursday"
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringFromTimeLocal()* on page 91, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringFromIntg()

Converts an integer value into its string representation in another base and returns the result.

Category

String

Syntax

```
SringResult = StringFromIntg( Number, numberBase );
```

Parameters

Number
Number to convert. Any number or an integer attribute.

numberBase
Base to use in conversion. Any number or an integer attribute.

Examples

```
StringFromIntg(26, 2); ' returns "11010"
StringFromIntg(26, 8); ' returns "32"
StringFromIntg(26, 16); ' returns "1A"
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringFromReal()

Converts a real value into its string representation, either as a floating-point number or in exponential notation, and returns the result.

Category

String

Syntax

```
StringResult = StringFromReal( Number, Precision, Type );
```

Parameters

Number
Converted to the *Precision* and *Type* specified. Any number or a float attribute.

Precision
Specifies how many decimal places is shown. Any number or an integer attribute.

Type
A string value that determines the display method. Possible values are:

f = Display in floating-point notation.

e = Display in exponential notation with a lowercase "e."

E = Display in exponential notation with an uppercase "E" followed by a plus sign and at least three exponential digits.

Examples

```
StringFromReal(263.355, 2,"f"); ' returns "263.36";
StringFromReal(263.355, 2,"e"); ' returns "2.63e2";
StringFromReal(263.355, 2,"E"); ' returns "2.63 E+002";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringFromTime()

Converts a time value (in seconds since January 1, 1970) into a particular string representation and returns the result.

Category

String

Syntax

```
StringResult = StringFromTime( SecsSince1-1-70, StringType );
```

Parameters

SecsSince1-1-70
Converted to the *StringType* specified.

StringType
Determines the display method. Possible values are:

1 = Shows the date in the same format set from the Windows Control Panel.

2 = Shows the time in the same format set from the Windows Control Panel.

3 = Shows a 24-character string indicating both the date and time: "Wed Jan 02 02:03:55 1993"

4 = Shows the short form for a day of the week: "Wed"

5 = Shows the long form for a day of the week: "Wednesday"

Remarks

The time value is UTC equivalent: number of elapsed seconds since January 1, 1970 GMT. The value returned reflects the local time.

Examples

```
StringFromTime(86400, 1); ' returns "1/2/1970"
StringFromTime(86400, 2); ' returns "12:00:00 AM"
StringFromTime(86400, 3); ' returns "Fri Jan 02 00:00:00 1970"
StringFromTime(86400, 4); ' returns "Fri"
StringFromTime(86400, 5); ' returns "Friday"
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringFromTimeLocal()

Converts a time value (in seconds since Jan-01-1970) into a particular string representation. The value returned also represents local time.

Category

String

Syntax

```
MessageResult=StringFromTimeLocal(SecsSince1-1-70,
```

StringType);

Parameters

SecsSince1-1-70
Is converted to the StringType specified and the result is stored in MessageResult.

StringType
Determines the display method:

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"

Remarks

Any adjustments necessary due to Daylight Savings Time will automatically be applied to the return result. Therefore, it is not necessary to make any manual adjustments for DST to the input value.

Example

```
StringFromTimeLocal (86400, 1); ' returns "1/2/1970"
StringFromTimeLocal (86400, 2); ' returns "12:00:00 AM"
StringFromTimeLocal (86400, 3); ' returns "Fri Jan 02 00:00:00 1970"
StringFromTimeLocal (86400, 4); ' returns "Fri"
StringFromTimeLocal (86400, 5); ' returns "Friday"
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringInString()

Returns the position in a string of text where a specified string first occurs.

Category

String

Syntax

```
IntegerResult = StringInString( Text, SearchFor, StartPos, CaseSens );
```

Parameters

Text
The string that is searched. Actual string or a string attribute.

SearchFor
The string to be searched for. Actual string or a string attribute.

StartPos
Determines the position in the text where the search begins. Any number or an integer attribute.

CaseSens
Determines whether the search is case-sensitive.

```
0 = Not case-sensitive
1 = Case-sensitive
Any number or an integer attribute.
```

Remarks

If multiple occurrences of *SearchFor* are found, the location of the first is returned.

Examples

```
StringInString("The mixer is running", "mix", 1, 0) ' returns 5;
StringInString("Today is Thursday", "day", 1, 0) ' returns 3;
StringInString("Today is Thursday", "day", 10, 0) ' returns 15;
StringInString("Today is Veteran's Day", "Day", 1, 1) ' returns 20;
StringInString("Today is Veteran's Day", "Night", 1, 1) ' returns 0;
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringLeft()

Returns a specified number of characters in a string value, starting with the leftmost string character.

Category

String

Syntax

```
StringResult = StringLeft( Text, Chars );
```

Parameters

Text
Actual string or a string attribute.

Chars
Number of characters to return or an integer attribute.

Remarks

If *Chars* is set to 0, the entire string is returned.

Examples

```
StringLeft("The Control Pump is On", 3) ' returns "The";
StringLeft("Pump 01 is On", 4) ' returns "Pump";
StringLeft("Pump 01 is On", 96) ' returns "Pump 01 is On";
StringLeft("The Control Pump is On", 0) ' returns "The Control Pump is On";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringLen()

Returns the number of characters in a string.

Category

String

Syntax

```
IntegerResult = StringLen( Text );
```

Parameter

Text
Actual string or a string attribute.

Remarks

All the characters in the string attribute are counted, including blank spaces and those not normally shown on the screen.

Examples

```
StringLen("Twelve percent") ' returns 14;
StringLen("12%") ' returns 3;
StringLen("The end." + StringChar(13)) ' returns 9;
```

The carriage return character is ASCII 13.

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringLower()

Converts all uppercase characters in text string to lowercase and returns the result.

Category

String

Syntax

```
StringResult = StringLower( Text );
```

Parameter

Text
String to be converted to lowercase. Actual string or a string attribute.

Remarks

Lowercase characters, symbols, numbers, and other special characters are not affected.

Examples

```
StringLower("TURBINE") ' returns "turbine";
StringLower("22.2 Is The Value") ' returns "22.2 is the value";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringMid()

Extracts a specific number of characters from a starting point within a string and returns the extracted character string as the result.

Category

String

Syntax

```
StringResult = StringMid( Text, StartChar, Chars );
```

Parameters

Text
Actual string or a string attribute to extract a range of characters.

StartChar
The position of the first character within the string to extract. Any number or an integer attribute.

Chars
The number of characters within the string to return. Any number or an integer attribute.

Remarks

This function is slightly different than the *StringLeft()* on page 92 function and *StringRight()* on page 95 function in that it allows you to specify both the start and end of the string that is to be extracted.

Examples

```
StringMid("The Furnace is Overheating",5,7);  ' returns "Furnace";
StringMid("The Furnace is Overheating",13,3);  ' returns "is ";
StringMid("The Furnace is Overheating",16,50); ' returns "Overheating"
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringReplace()

Replaces or changes specific parts of a provided string and returns the result.

Category

String

Syntax

```
StringResult = StringReplace( Text, SearchFor, ReplaceWith, CaseSens,
NumToReplace, MatchWholeWords );
```

Parameters

Text
The string in which characters, words, or phrases will be replaced. Actual string or a string attribute.

SearchFor
The string to search for and replace. Actual string or a string attribute.

ReplaceWith
The replacement string. Actual string or a string attribute.

CaseSens
Determines whether the search is case-sensitive. (0=no and 1=yes) Any number or an integer attribute.

NumToReplace
Determines the number of occurrences to replace. Any number or an integer attribute. To indicate all occurrences, set this value to -1.

MatchWholeWords
Determines whether the function limits its replacement to whole words. (0=no and 1=yes) Any number or an integer attribute. If *MatchWholeWords* is turned on (set to 1) and the *SearchFor* is set to "and", the "and" in "handle" are not replaced. If the *MatchWholeWords* is turned off (set to 0), it is replaced.

Remarks

Use this function to replace characters, words, or phrases within a string.

The *StringReplace()* on page 94 function does not recognize special characters, such as @ # $ % & * ( ). It reads them as delimiters. For example, if the function *StringReplace()* on page 94 (abc#,abc#,1234,0,1,1) is processed, there is no replacement. The # sign is read as a delimiter instead of a character.

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);
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringRight()

Returns the specified number of characters starting at the right-most character of text.

Category

String

Syntax

```
StringResult = StringRight( Text, Chars );
```

Parameters

Text
Actual string or a string attribute.

Chars
The number of characters to return or an integer attribute.

Remarks

If *Chars* is set to 0, the entire string is returned.

Examples

```
StringRight("The Pump is On", 2) ' returns "On";
StringRight("The Pump is On", 5) ' returns "is On";
StringRight("The Pump is On", 87) ' returns "The Pump is On";
StringRight("The Pump is On", 0) ' returns "The Pump is On";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringSpace()

Generates a string of spaces either within a string attribute or within an expression and returns the result.

Category

String

Syntax

```
StringResult = StringSpace( NumSpaces );
```

Parameter

NumSpaces
Number of spaces to return. Any number or an integer attribute.

Examples

All spaces are represented by the "×" character:

```
StringSpace(4) ' returns "××××";
"Pump" + StringSpace(1) + "Station" ' returns "Pump×Station";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringTest()

Tests the first character of text to determine whether it is of a certain type and returns the result.

Category

String

Syntax

```
DiscreteResult = StringTest( Text, TestType );
```

Parameters

Text
String that function acts on. Actual string or a string attribute.

TestType
Determines the type of test. Possible values are:

1 = Alphanumeric character ('A-Z', 'a-z' and '0-9')

2 = Numeric character ('0- 9')

3 = Alphabetic character ('A-Z' and 'a-z')

4 = Uppercase character ('A-Z')

5 = Lowercase 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() on page 97 function returns true to *DiscreteResult* if the first character in *Text* is of the type specified by *TestType*. Otherwise, false is returned. If the *StringTest()* on page 97 function contains more than one character, only the first character of the attribute is tested.

Examples

```
StringTest("ACB123",1) ' returns 1;
StringTest("ABC123",5) ' returns 0;
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringToIntg()

Converts the numeric value of a string to an integer value and returns the result.

Category

String

Syntax

```
IntegerResult = StringToIntg( Text );
```

Parameter

Text
String that function acts on. Actual string or a string attribute.

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, the string's value is equated to zero (0). Blank spaces are ignored. If the first character is a number, the system continues to read the subsequent characters until a non-numeric value is detected.

Examples

```
StringToIntg("ABCD"); ' returns 0;
StringToIntg("22.2 is the Value"); ' returns 22  (since integers are whole
numbers);
StringToIntg("The Value is 22"); ' returns 0;
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringToReal()

Converts the numeric value of a string to a real (floating point) value and returns the result.

Category

String

Syntax

```
RealResult = StringToReal( Text );
```

Parameter

Text
String that function acts on. Actual string or a string attribute.

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 encountered.

Examples

```
StringToReal("ABCD"); ' returns 0;
StringToReal("22.261 is the value"); ' returns 22.261;
StringToReal("The Value is 2"); ' returns 0;
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringTrim()* on page 99, *StringUpper()* on page 99, *Text()* on page 100

StringTrim()

Removes unwanted spaces from text and returns the result.

Category

String

Syntax

```
StringResult = StringTrim( Text, TrimType );
```

Parameter

Text
String that is trimmed of spaces. Actual string or a string attribute.

TrimType
Determines how the string is trimmed. Possible values are:

1 = Remove leading spaces to the left of the first non-space character

2 = Remove trailing spaces to the right of the last non-space character

3 = Remove all spaces except for single spaces between words

Remarks

The text is searched for white-spaces (ASCII 0x09-0x0D or 0x20) that are to be removed. *TrimType* determines the method used by the function:

Examples

All spaces are represented by the "×" character.

```
StringTrim("×××××This×is×a××test×××××", 1) ' returns "This×is×a××test×××××";
StringTrim("×××××This×is×a××test×××××", 2) ' returns "×××××This×is×a××test";
StringTrim("×××××This×is×a××test×××××", 3) ' returns "This×is×a×test";
```

The *StringReplace()* on page 94 function can remove ALL spaces from a specified a string attribute. Simply replace all the space characters with a "null."

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringUpper()* on page 99, *Text()* on page 100

StringUpper()

Converts all lowercase text characters to uppercase and returns the result.

Category

String

Syntax

```
StringResult = StringUpper( Text );
```

Parameter

Text
String to be converted to uppercase. Actual string or a string attribute.

Remarks

Uppercase characters, symbols, numbers, and other special characters are not affected.

Examples

```
StringUpper("abcd"); ' returns "ABCD";
StringUpper("22.2 is the value"); ' returns "22.2 IS THE VALUE";
```

See Also

StringASCII() on page 86, *StringChar()* on page 86, *StringFromIntg()* on page 89, *StringFromReal()* on page 89, *StringFromTime()* on page 90, *StringInString()* on page 92, *StringLeft()* on page 92, *StringLen()* on page 93, *StringLower()* on page 93, *StringMid()* on page 94, *StringReplace()* on page 94, *StringRight()* on page 95, *StringSpace()* on page 96, *StringTest()* on page 97, *StringToIntg()* on page 97, *StringToReal()* on page 98, *StringTrim()* on page 99, *Text()* on page 100

Text()

Converts a number to text based on a specified format.

Category

String

Syntax

```
StringResult = Text( Number, Format );
```

Parameters

Number
Any number or numeric attribute.

Format
Format to use in conversion. Actual string or a string attribute.

Examples

```
Text(66,"#.00"); ' returns 66.00;
Text(22.269,"#.00"); ' returns 22.27;
Text(9.999,"#.00"); ' returns 10.00;
```

The following example shows how to use this function within another function:

```
LogMessage("The current value of FreezerRoomTemp is:" + Text (FreezerRoomTemp,
"#.#"));
```

In the following example, MessageTag is set to "One=1 Two=2".

```
MessageTag = "One + " + Text(1,"#") + StringChar(32) +  "Two +" + Text(2,"#");
```

See Also

StringFromIntg() on page 89, *StringToIntg()* on page 97, *StringFromReal()* on page 89, *StringToReal()* on page 98

WWStringFromTime()

Converts a time value given in local time into UTC time (Coordinated Universal Time), and displays the result as a string.

Category

String

Syntax

```
MessageResult = wwStringFromTime(SecsSince1-1-70,StringType);
```

Parameters

SecsSince1-1-70
Integer Type. Number of Seconds elapsed since Jan 01 00:00:00 1970.

StringType
Determines the display method:

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"

Remarks

Any adjustments necessary due to Daylight Savings Time will automatically be applied to the return result. Therefore, it is not necessary to make any manual adjustments for DST to the input value.

Example

This example assumes that the time zone on the local node is Pacific Standard Time (UTC-0800). The local time passed to the function is 04:00:00 PM on Thursday, 1/1/1970. Since PST is 8 hours behind UTC, the function will return the following results:

```
wwStringFromTime(57600, 1) will return "1/2/70"
wwStringFromTime(57600, 2) will return "12:00:00 AM"
wwStringFromTime(57600, 3) will return "Fri Jan 02 00:00:00 1970"
wwStringFromTime(57600, 4) will return "Fri"
wwStringFromTime(57600, 5) will return "Friday"
```

System Functions

Use system functions to interact with the operating system or other core system functions, such as ActiveX objects.

CreateObject()

Creates an ActiveX (COM) object.

Category

System

Syntax

```
ObjectResult = CreateObject( ProgID );
```

Parameter

ProgID
The program ID (as a string) of the object to be created.

Example

```
CreateObject("ADODB.Connection");
```

Now()

Returns the current time.

Category

System

Syntax

```
TimeValue = Now();
```

Remarks

The return value can be formatted using .NET functions.

WWDDE Functions

Use WWDDE functions when working with the DDE protocol.

WWExecute()

Using the DDE protocol, executes a command to a specified application and topic and returns the status.

Category

WWDDE

Syntax

```
Status = WWExecute( Application, Topic, Command );
```

Parameters

Application
The application to which you want to send an execute command. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Command
The command to send. Actual string or a string attribute.

Return Value

Status is an Integer attribute to which 1, -1, or 0 is written. 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.

Remarks

Note: The three WWDDE functions Execute(), Poke() and Request() exist for legacy purposes.

The *Command* string is sent to a specified application and topic.

Important: The following applies to using WWExecute() in synchronous scripts:
1. Never loop them (call them over and over).
2. Never call several of them in a row and in the same script.
3. Never use them to call a lengthy task in another DDE application.
All three actions, though, are appropriate in asynchronous scripts.

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 is sent to Excel (and *TestMacro* runs):

```
[Run("Macro1!TestMacro")];
```

The following script executes a macro in Microsoft Access:

```
WWExecute("MSAccess","system","MyMacro");
```

WWPoke()

Using the DDE protocol, pokes a value to a specified application, topic, and item and returns the status.

Category

WWDDE

Syntax

```
Status = WWPoke( Application, Topic, Item, TextValue );
```

Parameters

Application
The application to which you want to send the Poke command. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Item
The item to poke within the topic. Actual string or a string attribute.

TextValue
The value to poke. If the value you want to send is a number, you can convert it using the *Text()* on page 100, *StringFromIntg()* on page 89, or *StringFromReal()* on page 89 functions. Actual string or a string attribute.

Return Value

Status is an Integer attribute to which 1, -1, or 0 is written. The WWPoke() function returns 1 if the application is running, the topic and item exist, and the value was sent successfully. It returns 0 if the application is busy, and -1 if there is an error.

Remarks

Note: The three WWDDE functions Execute(), Poke() and Request() exist for legacy purposes.

The value *TextValue* is sent to the particular application, topic, and item specified.

Important: The following applies to using WWRequest() in synchronous scripts:
1. Never loop them (call them over and over).
2. Never call several of them in a row and in the same script.
3. Never use them to call a lengthy task in another DDE application. All three actions, though, are appropriate in asynchronous scripts.

Example

The following statement converts a value to text and pokes the result to an Excel spreadsheet cell:

```
String=Text(Value,"0");
WWPoke("excel","[Book1.xls]sheet1","r1c1",String);
```

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.

See Also

Text() on page 100, *StringFromIntg()* on page 89, *StringFromReal()* on page 89

WWRequest()

Using the DDE protocol, makes a one-time request for a value from a particular application, topic, and item and returns the status.

Category

WWDDE

Syntax

```
Status = WWRequest( Application, Topic, Item, Attribute );
```

Parameters

Application
The application from which you want to request data. Actual string or a string attribute.

Topic
The topic within the application. Actual string or a string attribute.

Item
The item within the topic. Actual string or a string attribute.

Attribute
A string attribute, enclosed in quotation marks, that contains the requested value from the application, topic, and item. Actual string or a string attribute.

Return Value

Status is an integer attribute to which 1, -1, or 0 is written. The WWRequest() function returns 1 if the application is running, the topic and item exist, and the value was returned successfully. It returns 0 if the application is busy, and -1 if there is an error.

Remarks

Note: The three WWDDE functions Execute(), Poke() and Request() exist for legacy purposes.

The DDE value in the particular application, topic, and item is returned into *Attribute*.

The value is returned as a string into a string attribute. If the value is a number, you can then convert it using the *StringToIntg()* on page 97 or *StringToReal()* on page 98 functions.

Important: Never do the following when using WWRequest() in synchronous scripts:
1. Loop scripts (call them over and over).
2. Call several of scripts in a row and in the same script.
3. Use scripts to call a lengthy task in another DDE application.
All three actions can be done in asynchronous scripts.

Example

The following statement requests a value from an Excel spreadsheet cell and converts the resulting string into a value:

```
WWRequest("excel","[Book1.xls]sheet1","r1c1",Result);
Value=StringToReal(Result);
```

See Also

StringToIntg() on page 97, *StringToReal()* on page 98

QuickScript .NET Variables

QuickScript .NET variables must be declared before they can be used in QuickScript .NET scripts. Variables can be used on both the left and right side of statements and expressions.

Local variables or attributes can be used together in the same script. Variables declared within the script body lose their value after the script is executed. Those declared in the script body cannot be accessed by other scripts.

Variables declared in the **Declarations** area maintain their values throughout the lifetime of the object that the script is associated with.

Each variable must be declared in the script by a separate DIM statement followed by a semicolon. Enter DIM statements in the **Declarations** area of the **Script** tab page. The DIM statement syntax is as follows:

```
DIM <variable_name> [ ( <upper_bound>
[, <upper_bound >[, < upper_bound >]] ) ]
[ AS <data_type> ];
```

where:

DIM	**Required keyword.**
<variable_name>	Name that begins with a letter (A-Z or a-z) and whose remaining characters can be any combination of letters (A-Z or a-z), digits (0-9) and underscores (_). The variable name is limited to 255 Unicode characters.
<upper_bound>	Reference to the upper bound (a number between 1 and 2,147,483,647, inclusive) of an array dimension. Three dimensions are supported in a DIM statement, each being nested in the syntax structure. After the upper bound is specified, it is fixed after the declaration. A statement similar to Visual Basic's ReDim is not supported. The lower bound of each array dimension is always 1.
AS	Optional keyword for declaring the variable's datatype.

<data_type>	Any one of the following 11 datatypes: Boolean, Discrete, Integer, ElapsedTime, Float, Real, Double, String, Message, Time or Object.
	Data_type can also be a .Net data_type like System.Xml.XmlDocument or a type defined in an imported script library
	If you omit the AS clause from the DIM statement, the variable, by default, is declared as an Integer datatype. For example:
	`DIM LocVar1;`
	is equivalent to:
	`DIM LocVar1 AS Integer;`

In contrast to attribute names, variable names must not contain dots. Variable names and the data type identifiers are not case sensitive. If there is a naming conflict between a declared variable and another named entity in the script (for example, attribute name, alias or name of an object leveraged by the script), the variable name takes precedence over the other named entities. If the variable name is the same as an alias name, a warning message appears when the script is validated to indicate that the alias is ignored.

The syntax for specifying the entire array is "[ ]" for both local array variables and for attribute references. For example, to assign an attribute array to a local array, the syntax is:

```
locarr[] = tag.attr[];
```

DIM statements can be located anywhere in the script body, but they must precede the first referencing script statement or expression. If a local variable is referenced before the DIM statement, script validation done when you save the object containing the script prompts you to define it.

Caution: The validation mentioned above occurs only when you save the object containing the script. This is not the script syntax validation done when you click the **Validate Script** button.

Do not cascade DIM statements. For example, the following examples are invalid:

```
DIM LocVar1 AS Integer, LocVar2 AS Real;
DIM LocVar3, LocVar4, LocVar5, AS Message;
```

To declare multiple variables, you must enter separate `DIM` statements for each variable.

When used on the right side of an equation, declared local variables always cause expressions on the left side to have Good quality. For example :

```
dim x as integer;
dim y as integer;
x = 5;
y = 5;
me.attr = 5;
me.attr = x;
me.attr = x+y;
```

In each case of `me.attr`, quality is Good.

When you use a variable in an expression to the right of the operator, its Quality is treated as Good for the purpose of data quality propagation.

You can use null to indicate that there is no object currently assigned to a variable. Using null has the same meaning as the keyword "null" in C# or "nothing" in Visual Basic. Assigning null to a variable makes the variable eligible for garbage collection. You may not use a variable whose value is null. If you do, the script terminates and an error message appears in the logger. You may, however, test a variable for null. For example:

```
IF myvar == null THEN ...
```

It is not possible to pass attributes as parameters for system objects. To work around this issue, use a local variable as an intermediary or explicitly convert the attribute to a string using an appropriate function call when calling the system object.

Numbers and Strings

Allowed format for integer constants in decimal format is as follows:

```
IntegerConst =  0 or [sign] <non-zero_digit> <digit>*;
```

where:

```
sign :: = + | -
non-zero_digit ::= 1-9
digit ::= 0-9
```

For example, an integer constant is a zero or consists of an optional sign followed by one or more digits. Leading zeros are not allowed. Integer constants outside the range –2147483648 to 2147483647 cause an overflow error.

Prepending either 0x or 0X causes a literal integer constant to be interpreted as hexadecimal notation. The +/- sign is supported.

The acceptable float for integers in hexadecimal is as follows:

```
IntegerHexConst = [<sign>] <0><x (or X)> <hexdigit>*
```

where:

`sign` ::= + or -

`hexdigit` ::= 0-9, A-F, a-f (only eight hexdigits [32-bits] are allowed)

Allowed format for floats is as follows:

```
FloatConst ::= [<sign>] <digit>* .<digit>+ [<exponent>;]
```

or

```
[<sign>] <digit>+ [.<digit>* [<exponent>]];
```

where:

```
sign ::= + or -
digit ::= 0-9 (can be one or more decimal digits)
exponent = e (or E) followed by a sign and then digit(s)
```

Float constants are applicable as values for variables of type float, real, or double. For example, float constants do not take the number of bytes into account. Script validation detects an overflow when a float, real, or double variable has been assigned a float constant that exceeds the maximum value.

If no digits appear before the period (.), at least one must appear after it. If neither an exponent part nor the period appears, a period is assumed to follow the last digit in the string.

If an attribute reference exists that has a format similar to a float constant with an exponent (such as "5E3"), then use the Attribute qualifier, as follows:

```
Attribute("5E3")
```

Strings must be surrounded by double quotation marks. They are referred to as quoted strings. The double-double quote indicates a single double-quote in the string. For example, the string:

```
Joe said, "Look at that."
```

can be represented in QuickScript .NET as:

```
"Joe said, ""Look at that."""
```

QuickScript .NET Control Structures

QuickScript .NET provides five primary control structures in the scripting environment:

- *IF … THEN … ELSEIF … ELSE … ENDIF* on page 107
- *FOR … TO … STEP … NEXT Loop* on page 109
- *FOR EACH … IN … NEXT* on page 110
- *TRY … CATCH* on page 111
- *WHILE Loop* on page 112

IF … THEN … ELSEIF … ELSE … ENDIF

IF-THEN-ELSE-ENDIF conditionally executes various instructions based on the state of an expression. The syntax is as follows:

```
IF <Boolean_expression> THEN
    [statements];
[ { ELSEIF
    [statements] } ];
[ ELSE
    [statements] ];
ENDIF;
```

Where Boolean_expression is an expression that can be evaluated as a Boolean.

Depending on the data type returned by the expression, the expression is evaluated to constitute a True or False state according to the following table:

Data Type	Mapping
Boolean, Discrete	Directly used (no mapping needed).

Data Type	Mapping
Integer	Value = 0 evaluated as False. Value != 0 evaluated as True.
Float, Real	Value = 0 evaluated as False. Value != 0 evaluated as True.
Double	Value = 0 evaluated as False. Value != 0 evaluated as True.
String, Message	Cannot be mapped. Using an expression that results in a string type as the Boolean_expression results in a script validation error.
Time	Cannot be mapped. Using an expression that results in a time type as the Boolean_expression results in a script validation error.
ElapsedTime	Cannot be mapped. Using an expression that results in an elapsed time type as the Boolean_expression results in a script validation error.
Object	Using an expression that results in an object type. Validates, but at run time, the object is converted to a Boolean. If the type cannot be converted to a Boolean, a run-time exception is raised.

The first block of statements is executed if Boolean_expression evaluates to True. Optionally, a second block of statements can be defined after the keyword ELSE. This block is executed if the Boolean_expression evaluates to False.

To help decide between multiple alternatives, an optional ELSEIF clause can be used as often as needed. The ELSEIF clause mimics switch statements offered by other programming languages. For example:

```
IF value == 0 Then
          Message = "Value is zero";
     ELSEIF value > 0 Then
          Message = "Value is positive";     ELSEIF value < 0 Then
          Message = "Value is negative";
     ELSE

          {Default. Should never occur in this example};
ENDIF;
```

The following approach nests a second IF compound statement within a previous one and requires an additional ENDIF:

```
IF (X1 == 1) THEN
           X1 = 5;
  { ELSEIF <X1 == 2> THEN
           X1 = 10;
    ELSEIF X1 == 3 THEN
           X1 = 20 ;
    ELSEIF X1 == 4 THEN
           X1 = 30 };
        IF X1 == 99 THEN
           X1 = 0;
        ENDIF;
```

```
ENDIF;
```

See Sample Scripts for more ideas about using this type of control structure.

IF … THEN … ELSEIF … ELSE … ENDIF and Attribute Quality

When an attribute value is copied to another attribute of the same type, the attribute's quality is also copied. This can be especially relevant when working with I/O attributes. For example, the following two statements copy both value and quality:

```
me.Attr2 = me.Attr1;
me.Attr2.value = me.Attr1.value;
```

If only the value needs to be copied and the attribute has the quality BAD, you can use a temporary variable to hold the value. For example:

```
Dim temp as Integer;
temp = me.Attr1;
me.Attr2 = temp;
```

If there is a comparison such as Attr1 <> Attr2 and one of the attributes has the quality BAD, then the statements within the IF control block are not executed. For example, assuming Attr1 has the quality BAD:

```
if me.Attr1<> me.Attr2 then
     me.Attr2 =  me.Attr1;
endif;
```

In this script, the statement `me.Attr2 = me.Attr1` is not executed because Attr1 has the quality BAD and comparing a BAD quality value with a good quality value is not defined/not possible.

The recommended approach is to first verify the quality of Attr1, as shown in the following example:

```
if(IsBad(me.Attr1)) then
LogMessage("Attr1 quality is bad, its value is not copied to Attr2");
else
  if me.Attr1<> me.Attr2 then
      me.AttrA2 =  me.Attr1;
  endif;
endif;
```

An alternative method of verifying quality is to use the "==" operator:

```
if Me.Attr1 == TRUE then
```

Or, you can add the "value" property to the simplified IF THEN statement:

```
if Me.Attr1.value then
```

Using any of the above methods to verify data quality will ensure that your scripts execute correctly.

FOR … TO … STEP … NEXT Loop

FOR-NEXT performs a function (or set of functions) within a script several times during a single execution of a script. The general format of the FOR-NEXT loop is as follows:

```
FOR <analog_var> = <start_expression> TO <end_expression> [STEP
<change_expression>];
          [statements];
          [EXIT FOR;];
          [statements];
NEXT;
```

Where:

- analog_var is a variable of type Integer, Float, Real, or Double.
- start_expression is a valid expression to initialize analog_var to a value for execution of the loop.

- end_expression is a valid expression. If analog_var is greater than end_expression, execution of the script jumps to the statement immediately following the NEXT statement.

  This holds true if loop is incrementing up, otherwise, if loop is decrementing, loop termination occurs if analog_var is less than end_expression.

- change_expression is an expression that defines the increment or decrement value of analog_var after execution of the NEXT statement. The change_expression can be either positive or negative.
  - If change_expression is positive, start_expression must be less than or equal to end_expression or the statements in the loop do not execute.
  - If change_expression is negative, start_expression must be greater than or equal to end_expression for the body of the loop to be executed.
- If STEP is not set, then change_expression defaults to 1 for increasing increments, and defaults to -1 for decreasing increments.

Exit the loop from within the body of the loop with the EXIT FOR statement.

The FOR loop is executed as follows:

1. analog_var is set equal to start_expression.
2. If change_expression is positive, the system tests to see if analog_var is greater than end_expression. If so, the loop exits. If change_expression is negative, the system tests to see if analog_var is less than end_expression. If so, program execution exits the loop.
3. The statements in the body of the loop are executed. The loop can potentially be exited via the EXIT FOR statement.
4. analog_var is incremented by 1,-1, or by change_expression if it is specified.
5. Steps 2 through 4 are repeated.

Note: FOR-NEXT loops can be nested. The number of levels of nesting possible depends on memory and resource availability.

FOR EACH … IN … NEXT

FOR EACH loops can be used only with collections exposed by OLE Automation servers. A FOR-EACH loop performs a function (or set of functions) within a script several times during a single execution of a script. The general format of the FOR-EACH loop is as follows:

```
FOR EACH <object_variable> IN <collection_object >
   [statements];
   [EXIT FOR;];
   [statements];
NEXT;
```

Where:

- object_variable is a dimmed variable.
- collection_object is a variable holding a collection object.

As in the case of the FOR … TO loop, it is possible to exit the execution of the loop through the statement EXIT FOR from within the loop.

TRY ... CATCH

TRY ... CATCH provides a way to handle some or all possible errors that may occur in a given block of code, while still running rather than terminating the program. The TRY part of the code is known as the try block. Deal with any exceptions in the CATCH part of the code, known as the catch block.

The general format for TRY ... CATCH is as follows:

```
TRY
    [try statements] 'guarded section
CATCH
    [catch statements]
ENDTRY
```

Where:

tryStatements

Statement(s) where an error can occur. Can be a compound statement. The tryStatement is a guarded section.

catchStatements

Statement(s) to handle errors occurring in the associated Try block. Can be a compound statement.

Note: Statements inside the Catch block may reference the reserved ERROR variable, which is a .NET System.Exception thrown from the Try block. The statements in the Catch block run only if an exception is thrown from the Try block.

TRY ... CATCH is executed as follows:

1. Run-time error handling starts with TRY. Put code that might result in an error in the try block.
2. If no run-time error occurs, the script will run as usual. Catch block statements will be ignored.
3. If a run-time error occurs, the rest of the try block does not execute.
4. When a run-time error occurs, the program immediately jumps to the CATCH statement and executes the catch block.

   The simplest kind of exception handling is to stop the program, write out the exception message, and continue the program.

   The error variable is not a string, but a .NET object of System.Exception. This means you can determine the type of exception, even with a simple CATCH statement. Call the GetType() method to determine the exception type, and then perform the operation you want, similar to executing multiple catch blocks.

Example:

```
dim command = new System.Data.SqlClient.SqlCommand;
dim reader as System.Data.SqlClient.SqlDataReader;
command.Connection = new System.Data.SqlClient.SqlConnection;
try
    command.Connection.ConnectionString = "Integrated Security=SSPI";
    command.CommandText="select * from sys.databases";
    command.Connection.Open();
    reader = command.ExecuteReader();

    while reader.Read()
        me.name = reader.GetString(0);
        LogMessage(me.name);
    endWhile;
catch
    LogMessage(error);
```

```
endtry;
if reader <> null and not reader.IsClosed then
    reader.Close();
endif;
if command.Connection.State == System.Data.ConnectionState.Open then
    command.Connection.Close();
endif;
```

WHILE Loop

WHILE loop performs a function or set of functions within a script several times during a single execution of a script while a condition is true. The general format of the WHILE loop is as follows:

```
WHILE <Boolean_expression>
        [statements]
        [EXIT WHILE;]
        [statements]
ENDWHILE;
```

Where: Boolean_expression is an expression that can be evaluated as a Boolean as defined in the description of IF…THEN statements.

It is possible to exit the loop from the body of the loop through the EXIT WHILE statement.

The WHILE loop is executed as follows:

1. The script evaluates whether the Boolean_expression is true or not. If not, program execution exits the loop and continues after the ENDWHILE statement.
2. The statements in the body of the loop are executed. The loop can be exited through the EXIT WHILE statement.
3. Steps 1 through 2 are repeated.

Note: WHILE loops can be nested. The number of levels of nesting possible depends on memory and resource availability.

QuickScript .NET Operators

The following QuickScript .NET operators require a single operand:

Operator	Short Description
~	Complement
-	Negation
NOT	Logical NOT

The following QuickScript .NET operators require two operands:

Operator	Short Description
+	Addition and concatenation
-	Subtraction
&	Bitwise AND
*	Multiplication

Operator	Short Description
**	Power
/	Division
^	Exclusive OR
|	Inclusive OR
<	Less than
<=	Less than or equal to
<>	Not equal to
=	Assignment
==	Equivalency (is equivalent to); not supported for entire array compares. Arrays must be compared one element at a time using ==.
>	Greater than
>=	Greater than or equal to
AND	Logical AND
MOD	Modulo
OR	Logical OR
SHL	Left shift
SHR	Right shift

The following table shows the precedence of QuickScript .NET operators:

Precedence	Operator
1 (highest)	( )
2	- (negation), NOT, ~
3	**
4	*, /, MOD
5	+, - (subtraction)
6	SHL, SHR
7	<, >, <=, >=
8	==, <>
9	&
10	^

Precedence	Operator
11	|
12	=
13	AND
14 (lowest)	OR

The arguments of the listed operators can be numbers or attribute values. Putting parentheses around an argument is optional. Operator names are not case-sensitive.

Parentheses ( )

Parentheses specify the correct order of evaluation for the operator(s). They can also make a complex expression easier to read. Operator(s) in parentheses are evaluated first, preempting the other rules of precedence that apply in the absence of parentheses. If the precedence is in question or needs to be overridden, use parentheses.

In the example below, parentheses add B and C together before multiplying by D:

```
( B + C ) * D;
```

Negation (-)

Negation is an operator that acts on a single component. It converts a positive integer or real number into a negative number.

Complement (~)

This operator yields the one's complement of a 32-bit integer. It converts each zero-bit to a one-bit and each one-bit to a zero-bit. The one's complement operator is an operator that acts on a single component, and it accepts an integer operand.

Power (**)

The Power operator returns the result of a number (the base) raised to the power of a second number (the power). The base and the power can be any real or integer numbers, subject to the following restrictions:

- A zero base and a negative power are invalid.

  Example: "0 ** - 2" and "0 ** -2.5"

- A negative base and a fractional power are invalid.

  Example: "-2 ** 2.5" and "-2 ** -2.5"

- Invalid operands yield a zero result.

The result of the operation should not be so large or so small that it cannot be represented as a real number. Example:

```
1 ** 1 = 1.0
3 ** 2 = 9.0
10 ** 5 = 100,000.0
```

Multiplication (*), Division (/), Addition (+),Subtraction (-)

These binary operators perform basic mathematical operations. The plus (+) can also concatenate String datatypes.

For example, in the data change script below, each time the value of "Number" changes, "Setpoint" changes as well:

```
Number=1;
Setpoint.Name = "Setpoint" + Text(Number, "#" );
```

Where: The result is "`Setpoint1`."

Modulo (MOD)

MOD is a binary operator that divides an integer quantity to its left by an integer quantity to its right. The remainder of the quotient is the result of the MOD operation. Example:

```
97 MOD 8 yields 1
63 MOD 5 yields 3
```

Shift Left (SHL), Shift Right (SHR)

SHL and SHR are binary operators that use only integer operands. The binary content of the 32-bit word referenced by the quantity to the left of the operator is shifted (right or left) by the number of bit positions specified in the quantity to the right of the operator.

Bits shifted out of the word are lost. Bit positions vacated by the shift are zero-filled. The shift is an unsigned shift.

Bitwise AND (&)

A bitwise binary operator compares 32-bit integer words with each other, bit for bit. Typically, this operator masks a set of bits. The operation in this example "masks out" (sets to zero) the upper 24 bits of the 32-bit word. For example:

```
result = name & 0xff;
```

Exclusive OR (^) and Inclusive OR (|)

The ORs are bitwise logical operators compare 32-bit integer words to each other, bit for bit. The Exclusive OR compare the status of bits in corresponding locations. If the corresponding bits are the same, a zero is the result. If the corresponding bits differ, a one is the result. Example:

```
0 ^ 0 yields 0
0 ^ 1 yields 1
1 ^ 0 yields 1
1 ^ 1 yields 0
```

The Inclusive OR examines the corresponding bits for a one condition. If either bit is a one, the result is a one. Only when both corresponding bits are zeros is the result a zero. For example:

```
0 | 0 yields 0
0 | 1 yields 1
1 | 0 yields 1
1 | 1 yields 1
```

Assignment (=)

Assignment is a binary operator which accepts integer, real, or any type of operand. Each statement can contain only one assignment operator. Only one name can be on the left side of the assignment operator.

Read the equal sign (=) of the assignment operator as "is assigned to" or "is set to."

Note: Do not confuse the equal sign with the equivalency sign (==) used in comparisons.

Comparisons (<, >, <=, >=, ==, <>)

Comparisons in IF-THEN-ELSE statements execute various instructions based on the state of an expression.

AND, OR, and NOT

These operators work only on discrete attributes. If these operators are used on integers or real numbers, they are converted as follows:

- Real to Discrete: If real is 0.0, discrete is 0, otherwise discrete is 1.
- Integer to Discrete: If integer is 0, discrete is 0, otherwise discrete is 1.

If the statement is: "`Disc1 = Real1 AND Real2;`" and `Real1` is 23.7 and `Real2` is 0.0, `Disc1` has 0 assigned to it, since `Real1` is converted to 1 and `Real2` is converted to 0.

When assigning the floating-point result of a mathematical operation to an integer, Application Server rounds the value to the nearest integer instead of truncating it. This means that an operation like `IntAttr = 32/60` results in `IntAttr` having a value of 1, not 0. If truncation is needed, use the `Trunc()` function.

CHAPTER 3

Sample QuickScript .NET Scripts

This section includes sample scripts to help you to understand the QuickScript .NET scripting language.

Important Notes: The sample scripts provided with a number of the Application Server scripting functions should work as written in most Windows operating system and installed software environments, but might not work with all possible hardware, operating system, and software combinations. We recommend that you modify the example scripts as necessary to fit your specific environment.

Some sample scripts include references to public websites as examples. You may need to replace those URLs with a current and verified URLs.

Accessing an Excel Spreadsheet Using an Imported Type Library

The purpose of this script is to write data to an open Excel spreadsheet.

Before using this script, you must first:

1. Import the Microsoft Office Excel dll (**Microsoft.Office.Interop.Excel.dll**) to create the required namespace. From the **Galaxy** menu, select **Import**, then **Script Function Library**.

   Note: The Microsoft Office Excel dll file name and location may vary, depending on which version of Excel is installed.

2. Open the Excel spreadsheet you want to access. In the following sample script, the Excel file name and path are: C:\documents\sample.xlsx

```
dim app as object;
dim wb as object;
dim ws as object;

app = CreateObject("Excel.Application");
app.Visible=true;
wb = app.Workbooks.Open("C:\documents\sample.xlsx");
ws = wb.ActiveSheet;
ws.Range("A1").Value = 100;
ws.Range("A2").Value = 200;
ws.Range("A3").Value = "=A1*A2";
LogMessage(ws.Range("A3").Value);
wb.Visible=true;
wb.Close(false);
```

Accessing an Excel Spreadsheet Using CreateObject

```
dim app as object;
dim wb as object;
dim ws as object;
app = CreateObject("Excel.Application");
wb = app.Workbooks.Add();
```

```
ws = wb.ActiveSheet;
ws.Range.("A1").value = 20;
ws.Range.("A2").value = 30;
ws.Range.("A3").value = "=A1*A2";
LogMessage(ws.Range("A3").Value);
wb.Close(false);
```

Calling a Web Service to Get the Temperature for a Specified Zip Code

Note: This sample script includes a reference to a public website as an example. You may need to replace that URL with a current and verified URL.

```
' Requires input string uda me.zipcode and output float uda me.temperature.
' First, generate a wrapper for the web service (.Net SDK must be installed).
' To generate wrapper, run the following commands from the DOS prompt:
'  set path=%path%;C:\Program Files\Microsoft Visual Studio
.NET\FrameworkSDK\Bin
'  wsdl http://www.vbws.com/services/weatherretriever.asmx
'  csc /target:library WeatherRetriever.cs
' Next import the generated WeatherRetriever.dll library into your galaxy.
' Now write your script:
dim wr as WeatherRetriever;
wr = new WeatherRetriever;
me.temperature = wr.GetTemperature(me.zipcode);
```

Calling a Web Service to Send an E-mail Message

Note: This sample script includes a reference to a public website as an example. You may need to replace that URL with a current and verified URL.

```
' First, generate a wrapper for the web service (.Net SDK must be installed).
' To generate wrapper, run the following commands from the DOS prompt:
' set path=%path%;C:\Program Files\Microsoft Visual Studio
.NET\FrameworkSDK\Bin
' wsdl /namespace:SendMail
http://www.xml-webservices.net/services/messaging/smtp_mail/mailsender.asmx
' csc /target:library Message.cs
' Next import the generated Message.dll library into your galaxy.
' Now write your script:
dim m as SendMail.Message;
m = new SendMail.Message;
m.SendSimpleMail
(
{to:      } "<type valid email address here>",
{from:    } "<type valid email address here>",
{subject: } "Reminder to self",
{body:    } "Pick up eggs and milk on your way home."
 );
```

Creating a Look-up Table and Doing a Look-up on It

```
dim zipcodes as System.Collections.Hashtable;
zipcodes = new System.Collections.Hashtable;
zipcodes["Irvine"] = 92618;
zipcodes["Mission Viejo"] = 92692;
LogMessage(zipcodes["Irvine"]);
```

Creating an XML Document and Saving it to Disk

```
dim doc       as System.Xml.XmlDocument;
dim catalog   as System.Xml.XmlElement;
dim book      as System.Xml.XmlElement;
dim title     as System.Xml.XmlElement;
dim author    as System.Xml.XmlElement;
dim lastName  as System.Xml.XmlElement;
dim firstName as System.Xml.XmlElement;
' create new XML document rooted in catalog
doc = new System.Xml.XmlDocument;
catalog = doc.CreateElement("catalog");
doc.AppendChild(catalog);
' add a book to the catalog
book = doc.CreateElement("book");
title = doc.CreateElement("title");
author = doc.CreateElement("author");
lastName = doc.CreateElement("lastName");
firstName = doc.CreateElement("firstName");
author.AppendChild(lastName);
author.AppendChild(firstName);
book.AppendChild(title);
book.AppendChild(author);
catalog.AppendChild(book);
book.SetAttribute("isbn", "0385503822");
title.InnerText = "The Summons";
lastName.InnerText = "Grisham";
firstName.InnerText = "John";
' add another book
book = doc.CreateElement("book");
title = doc.CreateElement("title");
author = doc.CreateElement("author");
lastName = doc.CreateElement("lastName");
firstName = doc.CreateElement("firstName");
author.AppendChild(lastName);
author.AppendChild(firstName);
book.AppendChild(title);
book.AppendChild(author);
catalog.AppendChild(book);
book.SetAttribute("isbn", "044023722X");
title.InnerText = "A Painted House";
lastName.InnerText = "Grisham";
firstName.InnerText = "John";
' save the XML document to disk
doc.Save("c:\catalog.xml");
```

Executing a SQL Parameterized INSERT Command

```
dim connection as System.Data.SqlClient.SqlConnection;
dim command as System.Data.SqlClient.SqlCommand;
dim regionId as System.Data.SqlClient.SqlParameter;
dim regionDesc as System.Data.SqlClient.SqlParameter;
dim commandText as string;
connection = new
System.Data.SqlClient.SqlConnection("server=(local);uid=sa;database=northwin
d");
connection.Open();
```

```
commandText = "INSERT INTO Region (RegionID, RegionDescription) VALUES (@id,
@desc)";
command = new System.Data.SqlClient.SqlCommand(commandText, connection);
regionId = command.Parameters.Add("@id", System.Data.SqlDbType.Int, 4);
regionDesc = command.Parameters.Add("@desc", System.Data.SqlDbType.NChar, 50);
command.Prepare();
regionId.Value = 5;
regionDesc.Value = "Europe";
command.ExecuteNonQuery();
regionId.Value = 6;
regionDesc.Value = "South America";
command.ExecuteNonQuery();
connection.Close();
```

Filling a String Array and Using It

```
dim numbers[3] as string;
dim s as string;
numbers[1] = "one";
numbers[2] = "two";
numbers[3] = "three";
LogMessage(numbers[3]);
for each s in numbers[]
LogMessage(s);
next;
```

Filling a Two-Dimensional Integer Array and Using It

```
dim x[2,3] as integer;
dim i as integer;
x[1, 1] = 1;
x[1, 2] = 2;
x[1, 3] = 3;
x[2, 1] = 4;
x[2, 2] = 5;
x[2, 3] = 6;
LogMessage(x[2, 3]);
for each i in x[]
LogMessage(i);
next;
```

Formatting a Number Using a .NET Format 'Picture'

```
dim i as integer;
i = 1234;
LogMessage("Total cost: " + i.ToString("$#,###,###.00"));
```

Formatting a Time Using a .NET Format 'Picture'

```
dim t as time;
t = Now();
LogMessage("The current time is: " + t.ToString("hh:mm:ss") + ".");
```

Getting the Directories Under the C Drive

```
dim dir as System.IO.DirectoryInfo;
for each dir in System.IO.DirectoryInfo("c:\").GetDirectories()
LogMessage(dir.FullName);
next;
```

Loading an XML Document from Disk and Doing Look-ups on It

```
dim doc as System.Xml.XmlDocument;
dim node as System.Xml.XmlNode;
doc = new System.Xml.XmlDocument;
doc.Load("c:\catalog.xml");
' find the title of the book whose isbn is 044023722X
node = doc.SelectSingleNode("/catalog/book[@isbn='044023722X']/title");
LogMessage(node.InnerText);
'  find all titles written by Grisham
for each node in
doc.SelectNodes("/catalog/book[author/lastName='Grisham']/title")
    LogMessage(node.InnerText);
next;
```

Querying a SQL Server Database

```
dim connection as System.Data.SqlClient.SqlConnection;
dim command as System.Data.SqlClient.SqlCommand;
dim reader as System.Data.SqlClient.SqlDataReader;
connection = new
System.Data.SqlClient.SqlConnection("server=(local);uid=sa;database=northwin
d");
connection.Open();
command = new System.Data.SqlClient.SqlCommand("select * from customers",
connection);
reader = command.ExecuteReader();
while reader.Read()
    LogMessage(reader("CompanyName"));
endwhile;
reader.Close();
connection.Close();
```

Reading a Performance Counter

```
' Requires output float UDA me.PercentProcessorTime.
' Declarations
dim counter as System.Diagnostics.PerformanceCounter;
' Startup
counter = new System.Diagnostics.PerformanceCounter;
counter.CategoryName = "Processor";
counter.CounterName = "% Processor Time";
counter.InstanceName = "0";
' Execute
me.PercentProcessorTime = counter.NextValue();
```

Reading a Text File from Disk

```
dim sr as System.IO.StreamReader;
sr = System.IO.File.OpenText("c:\MyFile.txt");
while sr.Peek() > -1
    LogMessage(sr.ReadLine());
endwhile;
sr.Close();
```

Sharing a SQL Connection or Any Other .NET Object

In UserDefined_001 do this:

```
dim connection as System.Data.SqlClient.SqlConnection;
’ Startup
connection = new
System.Data.SqlClient.SqlConnection("server=(local);uid=sa;database=northwin
d");
connection.Open();
System.AppDomain.CurrentDomain.SetData
("NorthwindConnection", connection);
’ Shutdown
System.AppDomain.CurrentDomain.SetData("NorthwindConnection", Null);
connection.Close();
```

Then in UserDefined_002, UserDefined_003, and so on, do this:

```
dim connection as System.Data.SqlClient.SqlConnection;
connection = System.AppDomain.CurrentDomain.GetData
("NorthwindConnection");
if connection <> null then
System.Threading.Monitor.Enter(connection);
' use the connection
System.Threading.Monitor.Exit(connection);
endif;
```

Using DDE to Access an Excel Spreadsheet

```
WWPoke("excel", "sheet1", "r1c1", "Hello");
WWRequest("excel", "sheet1", "r1c1", me.Greeting);
' Note: use "" to embed double quotation marks in strings
WWExecute("excel", "sheet1",
"[SELECT(""R1C1"")][FONT.PROPERTIES(,""Bold"")]");
```

Using Microsoft Exchange to Send an E-mail Message

The following compatibility notes apply to use of this script:

- The script will not work on Microsoft Office 2010 or later due to changes in MAPI handling.
- The script is supported only for 32-bit versions of Microsoft Office.
- If you are using Microsoft Office 2007, you must download and install Microsoft Collaboration Data Objects (CDO) 1.2.1. Additional information on CDO 1.2.1 is available at: http://www.microsoft.com/en-us/download/details.aspx?id=3671

```
dim session as object;
dim msg as object;
dim sProfileInfo as string;
sProfileInfo = "<type valid Microsoft Exchange Server Name here>" +
StringChar(10) + "<type valid Exchange Server user name here>";
session = CreateObject("MAPI.Session");
session.Logon(, , False,False , , True, sProfileInfo);
msg = session.Outbox.Messages.Add();
msg.Recipients.Add("<type valid email address here>");
msg.Recipients.Resolve();
msg.Subject = "Reminder to self";
msg.Text = "Pick up eggs and milk on your way home.";
msg.Send();
session.Logoff();
```

Using Screen-Scraping to Get the Temperature for a City

Note: This sample script includes a reference to a public website as an example. You may need to replace that URL with a current and verified URL.

```
' Screen-scraping involves downloading a web page,
' then using a regular expression to retrieve the desired data.
' Requires input string UDA me.CityState, e.g. "Los Angeles,CA"
' and output float UDA me.temperature.
dim request as System.Net.WebRequest;
dim reader as System.IO.StreamReader;
dim regex as System.Text.RegularExpressions.Regex;
dim match as System.Text.RegularExpressions.Match;
request = System.Net.WebRequest.Create
(
"http://www.srh.noaa.gov/data/forecasts/zipcity.php?inputstring=" +
System.Web.HttpUtility.UrlEncode(me.CityState)
);
reader = new
System.IO.StreamReader(request.GetResponse().GetResponseStream());
regex = new System.Text.RegularExpressions.Regex("<br><br>(.*)&deg;F<br>");
match = regex.Match(reader.ReadToEnd());
me.temperature = match.Groups(1);
```

Using SMTP to Send an E-mail Message

```
System.Web.Mail.SmtpMail.Send
(
{from:    } "<type valid email address here>",
{to:      } "<type valid email address here>",
{subject: } "Reminder to self",
{body:    } "Pick up eggs and milk on your way home."

 );
```

Writing a Text File to Disk

```
dim sw as System.IO.StreamWriter;
sw = System.IO.File.CreateText("C:\MyFile.txt");
sw.WriteLine("one");
sw.WriteLine("two");
sw.WriteLine("three");
sw.Close();
```

Dynamically Binding an Indirect Variable to a Reference

You can dynamically bind a variable of type Indirect to an arbitrary reference string and then use it for get/set purposes. For example:

```
' Assume reference obj1.Attr1 has value of 7
dim x as indirect;
dim s as string;
s = "obj1.Attr1";
x.BindTo(s);      ' where s is any expression that returns a string.
    ' The string should be an ArchestrA reference.
obj2.Attr2 = x;   ' sets obj2.Attr2 to the reference x is bound to
    ' (obj1.Attr1 in this example, which has value of 7)
x = 1234; ' sets obj1.Attr1 (in this example) to 1234
```

```
IF WriteStatus(x) == MxStatusOk THEN
    ' ... do something
endif;
```

Note: You can use .BindTo with an attribute on another engine, but this requires additional scripting of the bind to ensure good quality. For more information, see *Binding to Off-engine Attributes* on page 124.

An unbound indirect returns no data.

If the Galaxy has Advanced Communication Management enabled, we do not recommend that you use references that are part of an ActiveOnDemand DIObject scan group in a script with an Indirect. The reference activation process is not in sync with the script execution, so using a function such as the IsUseable() function always returns false.

For example, the following scripting is NOT recommended.

In the declarations section:

```
Dim pPump as Indirect;
In Script Body section
pPump.BindTo("Pump_001.State"); 'Pump_001 is part of a DIObject scan group that
has ActiveOnDemand enabled
IF IsUsable(pPump)
THEN Do this….' this will not execute
ELSE
Do that…
ENDIF;
pPump.BindTo("Pump_002.State");'Pump_002 is part of a DIObject scan group that
has ActiveOnDemand enabled
IF IsUsable(pPump)
THEN Do this….' this will not execute
ELSE
Do that…
ENDIF;
```

In the script, only Pump_002 is executing all of the time.

Important: If you have an existing application that uses the same Indirect variable with scripting more than one time for the items extended to device integration (DI) items or for DI items directly, and you enable Advanced Communication Management in the IDE, these scripts behave differently or do not execute as expected.

Binding to Off-engine Attributes

Reference binding is inherently an asynchronous process. This means that a reference to an attribute hosted on the same engine is immediately available, but off-engine references can require additional scan cycles to bind. In scripting binding to off-engine attributes, we recommend you check indirect variables for quality before using them.

You can use the following guidelines when binding to off-engine attributes:

- Declare the indirect variables in the declarations section of the script. This retains the value across scan cycles.
- Implement a mechanism to define different states of the script to distinguish between the normal execution cycle and waiting for bound references to resolve.
- In the execution or assignment state, use BindTo() to bind the indirect variables. After BindTo, change the state to "waiting for references" to check these in the next scan.

- In the "waiting for reference" state, use IsGood() to check the quality of the indirect variables. When all references show good quality, change to the normal execution state. The variables then are usable.
- You can implement a TimeOut state if required. For example, the remote engine might be Off Scan.

The following script examples illustrate these guidelines.

```
dim bindedRef1 as indirect;
dim bindedRef2 as indirect;
dim scriptState as integer;

' Script States
' 0 = normal execution
' 1 = wait for remote references

if (scriptState==0) then;
   ' normal script logic
   ' perform normal tasks
   ' ...

   ' for this example, change the binding to remote engine
   if (System.DateTime.Now.Second mod 10 == 0) then;
      LogMessage("Change binding (1)");
      bindedRef1.BindTo("ApplicationObject_001.SimValue01");
      bindedRef2.BindTo("ApplicationObject_001.SimValue02");
      scriptState = 1;
   endif;
   if (System DateTime.Now.Second mod 10 == 5) then;
      LogMessage("Change binding (2)");
      bindedRef1.BindTo("ApplicationObject_001.SimValue2");
      bindedRef2.BindTo("ApplicationObject_001.SimValue1");
      scriptState = 1;
   endif;

endif;

if (scriptState==1) then;
   ' wait for remote references
   ' in this example we want two valid references

   if (IsGood(bindedRef1) and IsGood(bindedRef2)) then;
      LogMessage("Binded references are good.");
      scriptState = 0;
   endif;

endif;
```

As an alternative, you can use a WHILE-triggered script to allow evaluation at every scan cycle. Generally, WHILE loops are not recommended, but can be used to ensure execution.

```
In Declarations:

dim x as indirect;
dim y as boolean;

In Execute (while true: me.z):

if not y then
```

```
    'This could also be done in the startup script
    x.BindTo("Object1.attribute1");
    y = true;
endif;

if IsGood(x) then
    LogMessage(x);
    me.z = false;
endif;
```

Index

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W