Application Server Scripting Guide

Invensys Systems, Inc.

Revision B
Last Revision: 11/18/08

Copyright

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200

http://www.wonderware.com

For comments or suggestions about the product documentation, send an e-mail message to productdocs@wonderware.com.

Trademarks

Contents

Welcome

This guide explains how to write Application Server scripts. This guide does not explain programming concepts; rather it is a reference for you after you learned the basics of scripting in Application Server.

You should understand standard programming techniques before writing Application Server scripts. If you do not know how to program in any language, contact Wonderware or your distributor for information about training.

For more information on using Application Server, see the *Application Server User's Guide*.

You can view this document online or you can print it, in part or whole, in Adobe Reader.

Documentation Conventions

This documentation uses the following conventions:

Convention	Used for
Initial Capitals	Paths and file names.
Bold	Menus, commands, dialog box names, and dialog box options.
`Monospace`	Code samples and display text.

Technical Support

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

Before you contact Technical Support, refer to the relevant section(s) in this documentation for a possible solution to the problem. If you need to contact technical support for help, have the following information ready:

- The type and version of the operating system you are using.
- Details of how to recreate the problem.
- The exact wording of the error messages you saw.
- Any relevant output listing from the Log Viewer or any other diagnostic applications.
- Details of what you did to try to solve the problem(s) and your results.
- If known, the Wonderware Technical Support case number assigned to your problem, if this is an ongoing problem.

Chapter 1

Common Scripting Environment

This section describes common styles, syntax, commands, and behaviors of Application Server scripts.

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 74.

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 within Application Server.

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 or 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: Executes 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 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 or scripting done in the Startup and OnScan methods is also important because 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 in Startup or OnScan methods:

- Database access
- File system access, .csv, .xml, .txt, and so on
- Off-object referencing
- Dynamic referencing

Dynamic Referencing Considerations

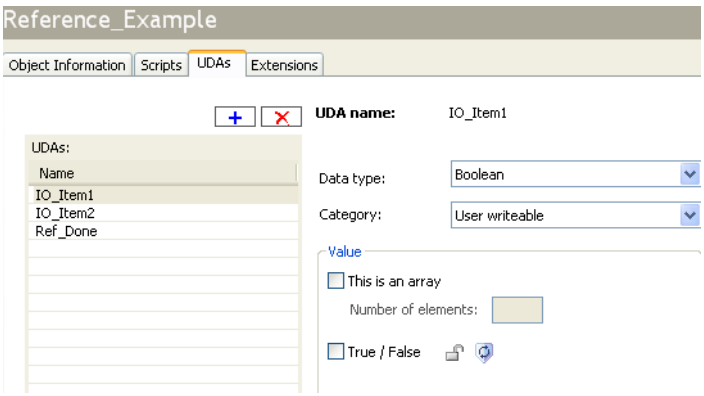
Dynamic reference scripting is one the biggest causes of deployment failures. It is one of the most common misuses of the Startup and OnScan methods.

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.

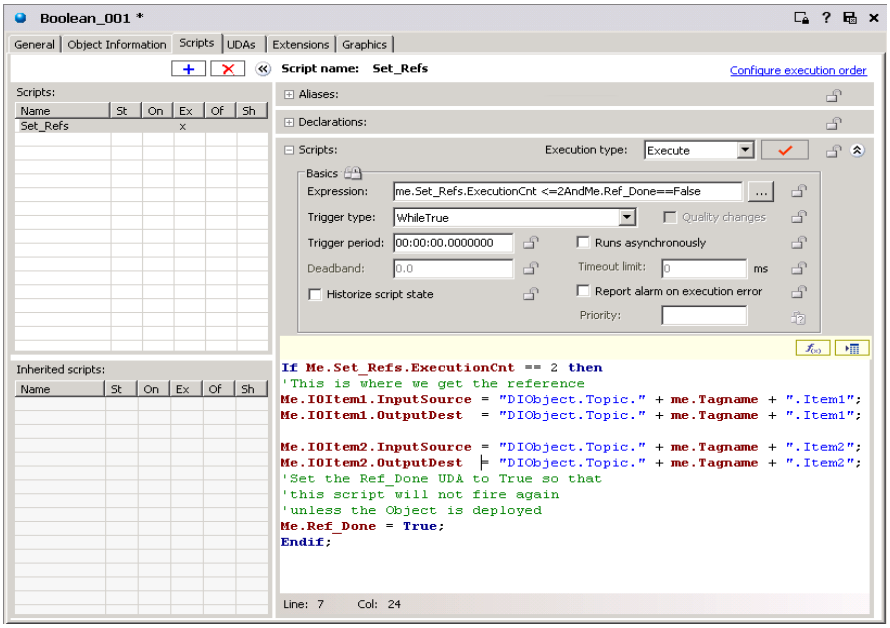
To create a simple dynamic reference script example

1 Create a Boolean UDA.



The UDA 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 UDAs, the `Ref_Done` UDA 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.

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	Dark blue
Comments (both single line and multi-line)	Green
Strings	Blue
Function names, numeric constants, operators, semicolons, dim variables, alias variables, and so on	Black
Attributes names	Maroon
Reserved words	Red

Chapter 2

QuickScript .NET Functions

This section describes the script functions included in the Application Server development environment.

Functions are listed alphabetically with:

- A description
- The function category, as shown in the Script Function Browser
- The proper syntax with descriptions of parameters
- Examples

An additional category of script functions shown in the Script Function Browser are the Types functions, which are not described in this documentation. The Types functions include .NET functions provided by the Microsoft .NET Framework and any .NET functions developed with Microsoft Visual Studio .NET.

For descriptions of each function provided by the Microsoft .NET Framework, see the Microsoft Developers Network website:

http://msdn.microsoft.com/

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

Script functions are described in this section.

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
```

ActivateApp()

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

Note Microsoft Vista operating system security prevents services from interacting with desktop applications. Object scripts that include the ActivateApp() function do not work when running under Vista. A warning message is written to the logger. But, the ActivateApp() function does work successfully with *client scripts* on computers running Vista.

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");
```

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(), Sin(), Tan(), ArcSin(), ArcTan()

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(), Sin(), Tan(), ArcCos(), ArcTan()

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(), Sin(), Tan(), ArcCos(), ArcSin()

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(), Tan(), ArcCos(), ArcSin(), ArcTan()

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");
```

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);
```

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");
```

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
```

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(), IsInitializing(), IsUncertain(), IsUsable()

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(), IsInitializing(), IsUncertain(), IsUsable()

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(), IsGood(), IsUncertain(), IsUsable()

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(), IsGood(), IsInitializing(), IsUsable()

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(), IsGood(), IsInitializing(), IsUncertain()

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(), Log10()

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(), Log10()

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(), LogN()

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);
```

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,
  "#"));
```

Now()

Returns the current time.

Category
System

Syntax

```
TimeValue = Now();
```

Remarks
The return value can be formatted using .NET functions.

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()

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.

Caution Microsoft Vista operating system security prevents services from interacting with desktop applications.Object scripts that include the SendKeys() function do not work when running under Vista. A warning message is written to the logger. But, the SendKeys() function does work successfully with *client scripts* on computers running Vista.

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.

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

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);
```

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(), SetInitializing(), SetUncertain()

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(), SetInitializing(), SetUncertain()

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(), SetGood(), SetUncertain()

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(), SetGood(), SetInitializing()

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(), Tan(), ArcCos(), ArcSin(), ArcTan()

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);
```

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;
```

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(), StringChar(), StringFromReal(), StringFromTime(), StringFromTimeLocal(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringCompareNoCase()

Compares a string value with another string and ignores the case.

Category
String

Syntax

```
StringCompareNoCompare( 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(), StringChar(), StringFromReal(), StringFromTime(), StringFromTimeLocal(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringFromTimeLocal(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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"
```

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."

Examples

```
StringFromReal(263.355, 2,"f"); ' returns
  "263.36";

StringFromReal(263.355, 2,"e"); ' returns
  "2.63e2";

StringFromReal(263.55, 3,"E"); ' returns
  "2.636E2";
```

See Also

StringASCII(), StringChar(), StringFromIntg(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringFromTimeLocal()

Converts a time value (in seconds since Jan-01-1970) into a particular string represention. 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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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() function and StringRight() 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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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() function does not recognize special characters, such as @ # $ % & * ( ). It reads them as delimiters. For example, if the function StringReplace() (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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

StringRight()

Returns the specified number of characters starting at the rightmost 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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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() 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() 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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringToIntg(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToReal(), StringTrim(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringTrim(), StringUpper(), Text()

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() function can remove ALL spaces from a specified a string attribute. Simply replace all the space characters with a "null."

See Also
StringASCII(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringUpper(), Text()

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(), StringChar(), StringFromIntg(), StringFromReal(), StringFromTime(), StringInString(), StringLeft(), StringLen(), StringLower(), StringMid(), StringReplace(), StringRight(), StringSpace(), StringTest(), StringToIntg(), StringToReal(), StringTrim(), Text()

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(), Sin(), ArcCos(), ArcSin(), ArcTan()

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(), StringToIntg(), StringFromReal(), StringToReal()

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()

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()

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(), StringFromIntg(), or StringFromReal() 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(), StringFromIntg(), StringFromReal()

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() or StringToReal() 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(), StringToReal()

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"
```

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 thoughout 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 UDAs as parameters for system objects. To work around this issue, use a local variable as an intermediary or explicitly convert the UDA 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 four primary control structures in the scripting environment:

- IF … THEN … ELSEIF … ELSE … ENDIF
- FOR … TO … STEP … NEXT Loop
- FOR EACH … IN … NEXT
- WHILE Loop

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).
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.

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 syntax is also supported:

```
IF <Boolean_expression> THEN;
           [statements];
[ { ELSEIF;
           [statements] } ];
[    ELSE;
           [statements] ];
     ENDIF;
ENDIF;
```

This approach nests a brand new IF compound statement within a previous one and requires an additional ENDIF.

See Sample Scripts on page 85 for more ideas about using this type of control structure.

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, to define 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.

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. 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 - 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.

See Sample Scripts on page 85 for ideas about using this type of control structure.

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.

See Sample Scripts on page 85 for ideas about using this type of control structure.

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.

See Sample Scripts on page 85 for ideas about using this type of control structure.

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
**	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

Precedence of operators are shown below:

Precedence	Operator
1 (highest)	( )
2	- (negation), NOT, ~
3	**
4	*, /, MOD
5	+, - (subtraction)
6	SHL, SHR
7	<, >, <=, >=
8	==, <>
9	&
10	^
11	|
12	=
13	AND
14 (lowest)	OR

Arguments for the previously listed operators can be numbers or attribute values. Putting parentheses around the arguments is optional. The 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.

Caution These sample scripts show proper syntax and structure. They may depend on resources or system configuration settings to run properly.

Sample Scripts

The sample scripts include:

- Accessing an Excel Spreadsheet Using an Imported Type Library
- Accessing an Excel Spreadsheet Using CreateObject
- Accessing an Office XP Excel Spreadsheet Using an Imported Type Library
- Calling a Web Service to Get the Temperature for a Specified Zip Code
- Calling a Web Service to Send an E-mail Message
- Creating a Look-up Table and Doing a Look-up on It
- Creating an XML Document and Saving it to Disk
- Executing a SQL Parameterized INSERT Command
- Filling a String Array and Using It
- Filling a Two-Dimensional Integer Array and Using It
- Formatting a Number Using a .NET Format 'Picture'

- Formatting a Time Using a .NET Format 'Picture'
- Getting the Directories Under the C Drive
- Loading an XML Document from Disk and Doing Look-ups on It
- Querying a SQL Server Database
- Reading a Performance Counter
- Reading a Text File from Disk
- Sharing a SQL Connection or Any Other .NET Object
- Using DDE to Access an Excel Spreadsheet
- Using Microsoft Exchange to Send an E-mail Message
- Using SMTP to Send an E-mail Message
- Using Screen-Scraping to Get the Temperature for a City
- Creating a Look-up Table and Doing a Look-up on It
- Dynamically Binding an Indirect Variable to a Reference

Accessing an Excel Spreadsheet Using an Imported Type Library

```
dim app as Excel._Application;
dim wb as Excel._Workbook;
dim ws as Excel._WorkSheet;

app = new Excel.Application;

wb = app.Workbooks.Add();
ws = wb.ActiveSheet;
ws.get_Range("A1").Value = 1000;
ws.get_Range("A2").Value = 1000;
ws.get_Range("A3").Value = "=A1+A2";
LogMessage(ws.get_Range("A3").Value);
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") = 20;
ws.Range("A2") = 30;
ws.Range("A3") = "=A1*A2";
LogMessage(ws.Range("A3").Value);

wb.Close(false);
```

Accessing an Office XP Excel Spreadsheet Using an Imported Type Library

```
dim app as Excel.Application;
dim ws as Excel.Worksheet;
dim wb as Excel.Workbook;
dim a1 as Excel.Range;
dim a2 as Excel.Range;
dim a3 as Excel.Range;

app = new Excel._ExcelApplicationClass;
wb = app.ActiveWorkbook;
ws = app.ActiveSheet;

a1 = ws.Range("A1");
a2 = ws.Range("A2");
a3 = ws.Range("A3");
a1.Value = 1000;
a2.Value = 2000;
a3.Value = "=A1*A2";

LogMessage(a3.Value);
wb.Close(true, "c:\temp.xls", false);
```

Calling a Web Service to Get the Temperature for a Specified Zip Code

This exact script may not work with the example Web site. Make sure you specify a valid Web site URL in your script.

```
' 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

```
' 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/sm
  tp_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=northwind");
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='044023722
  X']/title");
LogMessage(node.InnerText);

'  find all titles written by Grisham
for each node in
  doc.SelectNodes("/catalog/book[author/lastName='Gris
  ham']/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=northwind");
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=northwind");
connection.Open();
System.AppDomain.CurrentDomain.SetData
  ("NorthwindConnection", connection);

' Shutdown
Remove("NorthwindConnection");
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

```
dim session as object;
dim msg as object;


session = CreateObject("MAPI.Session");
session.Logon("Employee");
msg = session.Outbox.Messages.Add();
msg.Recipients.Add("<type valid email address here>");
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

```
' 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?in
  putstring=" +
System.Web.HttpUtility.UrlEncode(me.CityState)
);
reader = new
  System.IO.StreamReader(request.GetResponse().GetResp
  onseStream());
regex = new
  System.Text.RegularExpressions.Regex("<br><br>(.*)&d
  eg;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;
```

You cannot use .BindTo with an Indirect local variable to an attribute on another engine.

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 above 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.

Glossary

This glossary defines common terms in the Application Server and ArchestrA architecture.

application

A collection of objects within a Galaxy Repository that performs an automation task. Synonymous with Galaxy. You can have one or more applications within a Galaxy Repository.

ApplicationEngine (AppEngine)

A real-time engine that hosts and executes the run-time logic contained within AutomationObjects.

ApplicationObject Toolkit

A programmer's tool that creates new ApplicationObject templates, including configuration and run-time implementations.

ApplicationObject

Represents some element of your application. This can include things automation process components like

- thermocouple
- pump
- motor
- valve
- reactor
- tank

or associated application components like

- function block
- PID loop
- Sequential Function Chart
- Ladder Logic program
- batch phase
- SPC data sheet).

Application Server

The name of the product inside FactorySuite that forms a central application backbone. It includes the Galaxy Repository, one IDE, a WinPlatform, an AppEngine, and a basic library of ApplicationObjects.

Area

A logical grouping of AutomationObjects that represents an area or unit of a plant. An area groups related AutomationObjects for alarm, history, and security purposes. It is represented by an Area AutomationObject.

assignment The designation of a host for an AutomationObject. For example, an AppEngine AutomationObject is assigned to a WinPlatform AutomationObject.

attribute An externally accessible data item of an AutomationObject.

attribute reference string An unambiguous text string that references an attribute of an AutomationObject.

AutomationObject A type of object that represents permanent things in your plant, such as hardware, software, or engines, as objects with user-designated, unique names within the Galaxy. AutomationObjects provide a standard way to create, name, download, execute, and monitor the represented component.

base template A root template at the top of a derivational hierarchy. Unlike other templates, a base template is not derived from another template but developed with the ApplicationObject Toolkit and imported into a Galaxy.

block read group A DAGroup triggered by the user or another object. It reads a block of data from the external data source and indicates the completion status.

block write group A DAGroup triggered by the user or another object after all the required data items are set. The block of data is then sent to the external data device. When the block write is complete, it indicates the completion status.

change log The history log tracking the life cycle activities of ArchestrA, such as object creation, check-in/check-out, deployment, save, rename, undeploy, undo checkout, override checkout and assignment.

check-in IDE operation for persisting changes to an object to the Galaxy Repository and for making a configured object available for other users to check-out and use.

check-out IDE operation for editing an object, making it unavailable for other users to check-out.

checkpoint The act of saving on disk the configuration, state, and all associated data necessary to support automatic restart of a running AutomationObject. The restarted object has the same configuration, state, and associated data as the last checkpoint image on disk.

contained name The name of an object as it exists within the context of its container. For example, a valve object with the TagName "Valve101" can be contained within a reactor and given the ContainedName "Inlet" within the context of its container. The ContainedName must be unique within the context of the containing object.

containment The concept of placing one or more AutomationObjects within another AutomationObject. This results in a collection of AutomationObjects organized in a hierarchy that matches the application model and allows for better naming and manipulation. After placed in another AutomationObject, the contained object takes on a new name in addition to its unique tagname, known as the HierarchicalName. This name includes the container name and its name within the context of its container. For example, a level transmitter called TIC101 can be placed within a container object called Reactor1 and given the name Level within it, resulting in the HierarchicalName Reactor1.Level.

DAGroup A data access group associated with DeviceIntegration objects. It defines how communications is achieved with external data sources. It can contain subscription, block read, and block write scan groups.

Data Access Server (DAServer) The server executable that interfaces with DINetwork Objects and DIDevice Objects in the ArchestrA environment or with any third-party client, using various client protocols including OPC, DDE and SuiteLink.

Data Access Server Toolkit (DAS Toolkit) A developer tool to build Data Access Servers (DAServers).

DAServer Manager The Microsoft Management Console (MMC) snap-in supplied by the DAServer that provides the required user interface for activation, configuration, and diagnosis of the DAServer.

deployment The operation of creating an AutomationObject on its target PC. Includes installing the necessary software, the object's configuration data, and starting the object up.

derivation The creation of a new template based on an existing template.

derived template Any template with a parent template.

DeviceIntegration object (DIObjects) An AutomationObject that represents the communication with external devices. DIObjects run on an AppEngine, and include DINetwork Objects and DIDevice Objects.

DIDevice Object	A representation of an actual external device (for example, a PLC or RTU) that is associated with a DINetwork Object.
DINetwork Object	A representation of a physical connection to a DIDevice Object by means of the Data Access Server.
event record	The data that is transferred about the system and logged when a defined event changes state. For example, an analog crosses its high level limit or an acknowledgement is made.
export	The act of generating a Package file (.aaPKG file extension) from persisted data in the Galaxy Database. You can import the resulting .aaPKG file into another Galaxy through the IDE import mechanism.
framework	The ArchestrA infrastructure consisting of a common set of services, components, and interfaces for creating and deploying AutomationObjects that collect, store, visualize, control, track, report, and analyze plant floor processes and information.
Galaxy database	The relational database containing all persistent configuration information for all objects in a Galaxy.
Galaxy Repository	The software sub-system consisting of one or more Galaxy Databases.
Galaxy	Your entire application. The complete ArchestrA system consisting of a single logical name space and a collection of WinPlatforms, AppEngines and objects. One or more networked PCs that constitute an automation system. It defines the name space that all components and objects live in and defines the common set of system level policies that all components and objects comply with.
hierarchical name	The fully qualified name of a contained object, including the container object's TagName. For example, a valve object with a contained name of “Inlet” within a reactor named “Reactor1” would have “Reactor1.Inlet” as the HierarchicalName. The valve object also has a unique TagName distinct from its HierarchicalName, such as Valve101.
historical storage system	The time series data storage system, that compresses and stores high volumes of time series data for latter retrieval. Application Server leverages InSQL as its historical storage system.
host	An AutomationObject to which other objects are assigned (for example, a WinPlatform is a host for an AppEngine).

import	Adding templates or instances to the Galaxy Database from an external file.
instance	A uniquely configured representation of an application component based on a template.
instantiation	The creation of a new object based on a corresponding template.
Integrated Development Environment (IDE)	Consists of various configuration editors to configure the total system, for general framework use and for the creation of your application.
Log Viewer	A Microsoft Management Console (MMC) snap-in that provides a user interface for viewing messages reported to the LogViewer.
Message Exchange	The object-to-object messaging system.
object	Any template or instance found in a Galaxy Database. A common characteristic of objects is they are stored as separate components in the Galaxy Repository.
off-scan	The state of an AutomationObject that indicates it is idle and not ready to execute its normal runtime processing.
on-scan	The state of an AutomationObject in which it is performing its normal runtime processing based on a configured schedule.
Package Definition File (.aaPDF)	The standard description file that contains the configuration data and implementation code for a base template. File extension is typically .aaPDF.
Package File (.aaPKG)	The result of the export function of the IDE. The description file that contains the configuration data and implementation code for a template and/or instance. File extension is typically .aaPKG.
PLC	Programmable logic controller.
properties	Data common to all attributes of objects, such as name, value, quality, and data type.
reference	A string that refers to an object or to data within one of its attributes.
scan group	A DAGroup that requires only the update interval defined, and the data is retrieved at the requested rate.
System Management Console (SMC)	The central runtime system administration/management user interface.

TagName The unique name given to an instance. For example, an object might have the following TagName: V1101.

template An object containing configuration information and, optionally, the code modules to create new objects, including derived templates and instances.

toolset A named collection of templates listed together in the IDE Template ToolBox.

UserDefined object An AutomationObject created from the $UserDefined template. This template does not have any application-specific attributes or logic. Therefore, you must define these attributes and associated logic.

WinPlatform object An object that represents a single computer in a Galaxy, consisting of a systemwide message exchange component, a set of basic services, the operating system, and the physical hardware. This object hosts all AppEngines.

Index

Numerics

A

B

C

D

E

F

T

U

V

W