

Experion PKS Server Scripting Reference

EPDOC-X129-en-431A
February 2015

Release 431

Document	Release	Issue	Date
EPDOC-X129-en-431A	431	0	February 2015

Disclaimer

This document contains Honeywell proprietary information. Information contained herein is to be used solely for the purpose submitted, and no part of this document or its contents shall be reproduced, published, or disclosed to a third party without the express permission of Honeywell International Sàrl.

While this information is presented in good faith and believed to be accurate, Honeywell disclaims the implied warranties of merchantability and fitness for a purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any direct, special, or consequential damages. The information and specifications in this document are subject to change without notice.

Copyright 2015 - Honeywell International Sàrl

Contents

Getting started with server scripting	7
Scripting basics	9
Script types	10
Periodic scripts	10
Library scripts	11
Script execution	12
Recommended maximum script size	12
Scripting language	13
Events	14
Timers	15
Collections	17
Script transfer in a redundant server system	18
Referencing objects	19
Specifying times and dates	20
Storing script-derived data	21
Creating and raising alarms	22
Comparing strings	23
Using enumerated point parameters	24
Using server scripting error handlers	25
Avoiding operator interaction	26
Launching and interacting with external applications	27
Server redundancy issues	28
Security considerations for custom applications	29
Using the Script Editor	31
Writing scripts for process points	32
Writing scripts for standard points	33
Writing scripts for flexible points	34
Writing scripts for alerts	35
Writing scripts for the server, library or periodic scripts	36
Periodic script properties	36
Writing scripts for reports	38
Script Editor basics	39
Script Editor basics specific to Station	40
Script Editor keyboard shortcuts	40
Testing and debugging scripts	43
Simulating events	44
Script debugging	45
Configuring Internet Explorer for debugging Station scripts	45
Configuring Microsoft Visual Studio to debug server scripts	45
Checking script progress	48
Interpreting the server log	49
Server scripting error messages	50
0x80004005 (E_FAIL)	50
0x80040801 (0x80040000 M4_BUSY_TRB)	50
0x80040802 (0x80040000 M4_ILLEGAL_LRN)	51

0x80041001 (E_ITF_IN_COLLECTION)	51
0x8004101E (E_ITF_OBJ_NOT_CONFIGURED)	51
0x80041002 (E_ITF_NOT_IN_COLLECTION)	51
0x80048212 (0x80040000 M4_PNT_ON_SCAN)	51
0x80048231 (0x80040000 M4_INV_POINT)	52
0x80048232 (0x80040000 M4_INV_PARAMETER)	52
0x800483B0	52
0x800482B2 (0x80040000 M4_PARAM_NAME_EXISTS)	52
0x800482B4 (0x80040000 M4_INV_AREA)	53
0x800483FA	53
0x80070005 (E_ACCESSDENIED)	53
0x80070057 (E_INVALIDARG)	53
Script engines	55
Assigning scripts to script engines	56
Script queues	57
Script engine time-outs	58
Server startup and shutdown	59
Configuring and managing script engines	61
Configuring and managing automatic script engines	62
Automatic script engine properties	62
Reenabling scripts assigned to automatic script engines	63
Configuring and managing manual script engines	64
Manual script engine properties	64
Reenabling scripts assigned to manual script engines	65
Configuring debugging options	66
Server Scripting Object Model reference	67
Overview of the Server Scripting Object Model	68
Objects	70
AlarmDetails object	70
Alert object	71
Alerts collection	72
Area object	72
Areas collection	73
EventInfo object	74
Param object	77
ParamBlock object	78
Params collection	79
Point object	79
Points collection	80
Report object	81
Reports collection	82
Script object	82
Server object	83
Methods	85
CommitValue method	85
CreateAlarmDetails method	86
CreateParamBlock method	87
CreateTimer method	87
FireScriptEvent method	88
GenerateAlarm method	89
GenerateMessage method	90
KillTimer method	91
LogMessage method	92

ModifyTimer method	92
Request method	93
RequestTask method	93
SendFileToBackup method	94
TimerExists method	95
Events	96
OnAcknowledge event	96
OnAlarm event	96
OnAlert event	97
OnAlertAcknowledge event	97
OnAlertNormal event	98
OnChange event	98
OnCompletion event	99
OnDayStart event	99
OnNormal event	99
OnOperChange event	100
OnShutdown event	100
OnStartup event	101
OnTimer event	101
Example server scripting scenarios	103
Copying example Script Editor scripts	105
Calling up a display on a particular Station	107
Capturing and storing data in a text file	108
Changing a report's frequency	109
Changing a set point parameter with a library function	110
Changing the ranges of related points	111
Checking a point's value after it has gone into alarm	113
Checking a value at regular intervals	115
Counting the number of pump starts	116
Disabling and enabling alarms for related points	117
E-mail response for unacknowledged alarms	118
Generating a report in response to an alert	119
Generating and e-mailing reports	120
Generating messages in Station	121
Journaling an event at regular intervals	122
Launching an external application	123
Performing calculations on a process variable	124
Performing the same calculation on a set of points	125
Raising an alarm in response to an alert	126
Raising an urgent alarm when related points go into alarm	127
Reading data from a text file	128
Requesting a task when two points are enabled	130
Sending a report to a particular Station	131
Sending data to other displays	132
Setting a point to a specific value at the start of each day	133
Silencing a siren without acknowledging the alarm	134
Using an auxiliary parameter to store calculated data	135
Using auxiliary parameters to store and display data	136
Working with server redundancy	137
Notices	141
Documentation feedback	142
How to report a security vulnerability	143
Support	144

Training classes 145

Getting started with server scripting


Server scripts allow you to extend the functionality of your system. (A script is a mini-program that performs a specific task.)

Revision history

Revision	Date	Description
A	February 2015	Initial release of document.

How to use this guide

For	Go to
Scripting basics	“Scripting basics” on page 9
Tips for testing and debugging your scripts	“Testing and debugging scripts” on page 43
An introduction to the Script Editor, the tool you use to write scripts	“Using the Script Editor” on page 31
A detailed description of each object, method and event in the Server Scripting Object Model	“Server Scripting Object Model reference” on page 67
Example scenarios which include scripts that perform practical tasks	“Example server scripting scenarios” on page 103

 **Attention**
If you want to write a script that requires operator interaction or animates a display, you must write a *display script* (a display-based script). See the *HMIWeb Display Building Guide*.

Scripting basics

This section describes the basic rules and guidelines applicable to writing server scripts.

When considering the use of server scripts, and before you start any script writing, you should remain aware of the following server scripting restrictions and warnings:

- Do not use server scripts for plant or process control. All control should be handled by equipment or process controllers, not by server scripts.
- Do not use server scripts for manipulation of displays. All display script should occur within the display, and not on the server.
- Do not use server scripts for computationally intensive tasks or extensive file handling. For these operations, you should write a custom application in Visual C/C++ or Visual Basic.
- Server scripting should only be configured by competent systems engineers who have at least a rudimentary understanding of OOP principles and VBS. (If you don't know what these mean, then you shouldn't be attempting to write a script. You're more likely than not going to need some assistance.)
- Server scripting cannot involve operator interaction (because server scripts run in a desktop that is not visible to users). If you want operator interaction, you must use a display script.
- The scripting subsystem does not handle the state of scripts if a failover occurs.
- Server script scope is limited to using only the 'Server Scripting Object' model. To access the Server API and Network API functionality, you should instead use custom applications and utilities.
- Server scripting has been optimized for relatively short scripts (less than 30 lines), and doesn't block point processing or impact other server functions, because server scripts run at a low priority.
- All server scripts must be thoroughly tested and debugged before being implemented into your system.

A script is a mini-program that performs a specific task, which allows you to extend the automated functionality of your system. A server script can be written to extend the functionality of a process point, a standard point, a flexible point, an alert, a report, or the server itself.

Server scripts associated with particular objects are *event-driven*, which means that a script only runs when the associated event occurs—for example, when:

- A point changes state
- An operator acknowledges an alarm
- The server starts
- A report is generated

Server scripts can also include:

- *Periodic* scripts, which are not associated with a particular object or event, and run at specified intervals while the server is running
- *Library* scripts, which perform specialized functions when called by other server scripts

Script types

You can write the following types of server scripts.

Script type	Description
Point	<p>The script is attached to an event for a point or a point parameter—for example, the OnAlarm event for a point.</p> <p>You can write scripts for the following point types:</p> <ul style="list-style-type: none"> • Process (Process Controller point) • Flexible (system interface/point server point) • Standard
Server	The script is attached to an event for the server—for example, the OnStartup event.
Report	The script is attached to an event for a report—for example, the OnCompletion event.
Alert	The script is attached to an event for an alert—for example, the OnAlert event.
Periodic	The script runs at regular intervals while the server is running.
Library	<p>The script only runs when it is called by another script.</p> <p>As the name implies, library scripts are typically used to perform standard tasks. For example, if you need to perform the same calculation on numerous point parameters, you would use a library script to perform the calculation, and call it from each parameter script.</p>

Notes

- You cannot write scripts for the following items:
 - Alarm groups
 - Assets
 - System
- If you want to write a script that requires operator interaction or animates a display, you must write a *display script* (a display-based script). See HMIWeb Display Builder's help.

Periodic scripts

A *periodic script* runs at a regular intervals, without being attached to a particular object. For example, you could use a periodic script to journal the current values of certain points at hourly intervals.

In addition to the script itself, a periodic script has several properties, including the interval and the date/time at which the script runs for the first time.

Notes

- Periodic scripts cannot be called by other scripts—however, periodic scripts can call Library scripts.
- If you need to run scripts at regular intervals but want to associate them with particular objects, use Timers.
- You can have a maximum of 250 periodic scripts.

Related topics

“Journaling an event at regular intervals” on page 122

“Library scripts” on page 11

Library scripts

Library scripts are typically used to perform standard tasks. For example, if you need to perform the same calculation on numerous point parameters, you would use a library script to perform the calculation, and call it from the parameter scripts.

Unlike point, server and report scripts, library scripts are not attached to events, and therefore never run by themselves.

Library scripts have two types of syntax, either:

```
Sub SubName (arg1, ..., argn)
.
.
End Sub
```

or:

```
Function FunctionName (arg1, ..., argn)
.
.
End Function
```

Calling library scripts

For subroutines, either:

```
Call Subname (arg1, ..., argn)
```

or:

```
Subname arg1, ..., argn
```

For functions:

```
FunctionResult = Functionname (arg1, ..., argn)
```

Notes

- Library scripts can call other library scripts.
- The *Sub/End Sub* and *Function/End Function* statements are not hidden by the Script Editor in Library scripts.
- A function can be called using the subroutine calling syntaxes. However, if the subroutine calling syntaxes are used, the return value from the function will not be accessible.

Related topics

“Periodic scripts” on page 10

“Timers” on page 15

“Journaling an event at regular intervals” on page 122

Script execution

When server scripts run, they:

- Do not impact other server functions because they run at a low priority
- Do not block point processing because the associated events fire asynchronously into the Server Object Model

Recommended maximum script size

Server scripting has been optimized for relatively short scripts (less than 30 lines), and is not designed for implementing control strategies (these should be done in the controller).

**Attention**

You should not use a server script to perform a task if it is computationally intensive, or requires extensive file handling. Instead, you should write a custom application in Visual C/C++ or Visual Basic. For details about writing such an application, see the *Application Development Guide*.

Scripting language

You write scripts in VBScript, a popular scripting language that is easy to use and does not require extensive programming skills.

Script components

A script includes the following basic components:

- **Grammar**, such as *If ... Then ... End If*, which is provided by VBScript
- **Nouns** (objects) and **verbs** (methods and events), which are provided by the object model

```
Dim AlmDtls
Set AlmDtls = CreateAlarmDetails
If StrComp(ParamValue("SHUTDOWNSTATUS.STATE"), _
    "SHUTDOWN", vbTextCompare) = 0 Then
    'This does something
Else
    'This does something else
End If
```

Figure 1: Typical script (grammar is shown in blue)

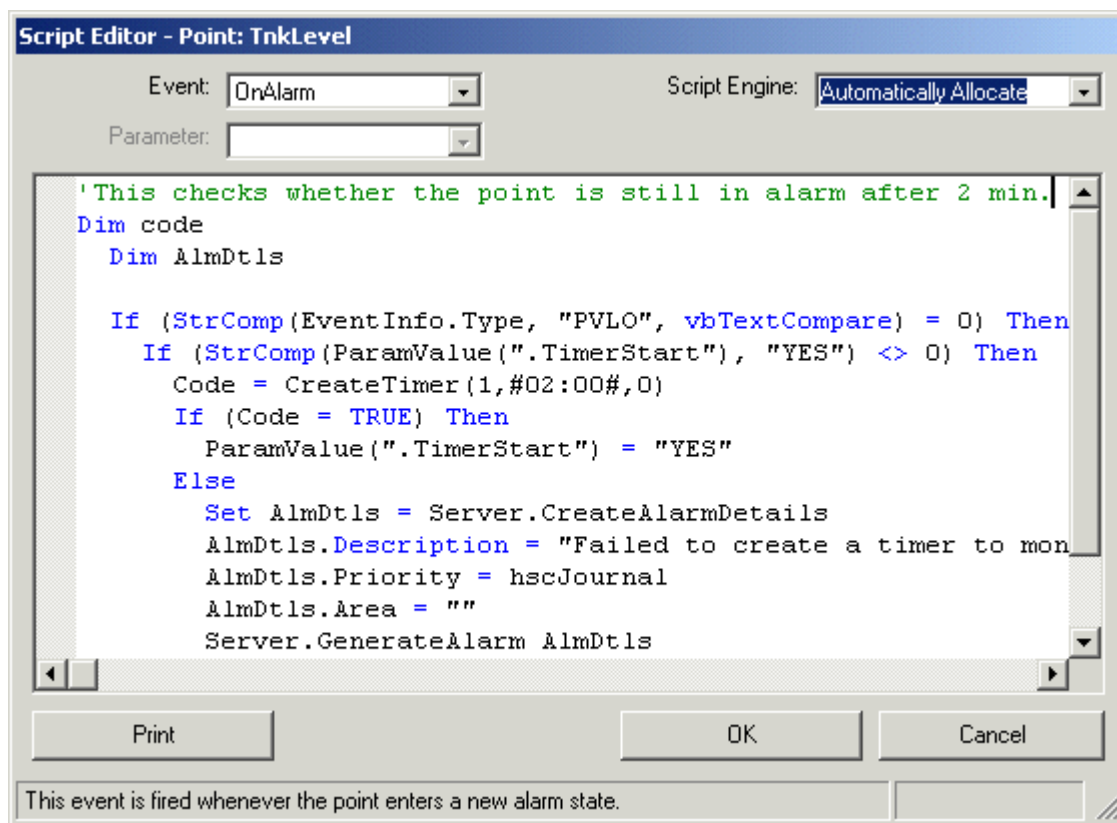
Help for VBScript

This help only describes the server object model (nouns and verbs). So, if you want help on VBScript's grammar, see its help. (The way in which you call up the help depends on the application in which you are writing the script.)

Events

Scripts (other than library scripts) are *event-driven*, which means that a script only runs when the associated event occurs.

The following figure shows a typical 'OnAlarm' script that runs when the point called 'TnkLevel' goes into alarm.



Notes

- Most events are only applicable to one type of object. For example, the OnAlarm event only applies to points.
- You can only write a script for an event in the 'associated' application. For example, you can only write an:
 - OnDayStart script in Station
 - OnAlarm script for a standard point in Quick Builder

Timers

A timer can only be created for one of the following objects:

- Point object
- Report object
- Server object

A *timer* fires OnTimer events for a specific object at regular intervals. A timer therefore enables you to run a script for an object at regular intervals—for example, you could generate a report for a particular point at 15-minute intervals.

Notes

- You create a timer with the CreateTimer method.
- You can change a timer's settings, such as its period, with the ModifyTimer method.
- You can check whether a timer exists with the TimerExists method.
- If you no longer need a timer, you kill it with the KillTimer method. (A timer is also killed if the server shuts down.)
- An object can have several timers. If it does, you must use the EventInfo object and its TimerID property to determine which timer fired the event. This allows one script to perform several tasks, each of which is associated with a particular timer. (The following example shows how to do this.)
- If you need to run scripts at regular intervals, but do not want to associate them with a particular object or event, use Periodic scripts.
- OnTimer events are only applicable to the application that created it. For example, if two applications create a timer for the Server object with TimerID of 10, but one application creates it with a 1 minute interval and the other creates it with a 5 minute interval, both these timers will co-exist and the respective applications will receive OnTimer events at the expected periods.

Example

This example shows how to perform a specialized task on a point called 'Pump5' at 15-minute intervals. To do this you:

- Create a timer for the point with the CreateTimer method. You can do this in any convenient script—in this example, the server's OnStartup event script is used. (In a real system, the OnStartup script is typically used to create and initialize many things.)
- Write the script, which performs the task, for the point's OnTimer event.

The server's OnStartup script creates the timer—with an ID of '2'—for the point. (Note that the time is specified in seconds, not minutes.)

```
Sub Server_OnStartup()
Points("Pump5").CreateTimer 2,900,0
'Any other startup tasks
.
.
End Sub
```

Because the point has more than one timer, its OnTimer script uses the EventInfo object to determine which timer triggered the event.

```
Sub Pump5_OnTimer()
If (EventInfo.TimerID = 1) Then
'perform task associated with timer 1
.
.
ElseIf (EventInfo.TimerID = 2) Then
'perform task associated with timer 2
.
End IfEnd Sub
```

If the point only had one timer, the script would be simpler because you would not need to use the EventInfo object to check the timer's ID.

```
Sub Pump5_OnTimer()  
'perform task  
.  
.  
End Sub
```

Example scenarios

“Changing a report's frequency” on page 109

“Silencing a siren without acknowledging the alarm” on page 134

“Checking a point's value after it has gone into alarm” on page 113

“E-mail response for unacknowledged alarms” on page 118

Related topics

“Journaling an event at regular intervals” on page 122

“Library scripts” on page 11

“CreateTimer method” on page 87

“Specifying times and dates” on page 20

“KillTimer method” on page 91

“ModifyTimer method” on page 92

“TimerExists method” on page 95

“OnTimer event” on page 101

Collections

A *collection* is a high-level object that represents all objects of the same type. Collections are typically used to select a particular object, or to perform an iterative task on a range of objects.

The object model contains the following collections:

- Alerts collection
- Areas collection
- Params collection
- Points collection
- Reports collection

Selecting an object

This example selects and generates (requests) the tenth report in the Reports collection.

```
Reports(10).Request
```

The Reports display shows the number assigned to each report.

Enumerations

Only the Areas and Reports collections support enumeration (which means the reports and areas can be counted, for example Reports (1), Reports(2) etc.). As such, they are only collections that support the *For ... Next* construct used in iterative tasks.

This example shows how to obtain the number of unacknowledged alarms for each assignable asset defined in your system—the script could then, for example, write this value to a file.

The asset detail display shows the area number for each asset.

```
For Each Area In Areas
AreaName = Area.Description
AlarmCount = Area.AlarmCount(hscAlarmCountAllUnack)
.
.
Next
```

Script transfer in a redundant server system

In a redundant server system, the scripting subsystem automatically transfers new or updated scripts to the backup server. (However, the scripts do not run on the backup server until it becomes the primary server.) It also deletes scripts from the backup server if they are deleted from the primary server.

The scripting subsystem does not specifically handle the state of scripts if a failover occurs. During a failover:

- All events queued to script engines on the old primary, but which have not run, are lost
- Only scripts attached to events that occur after the new primary takes over are run
- Scripts that were disabled because of a timeout or script error are re-enabled

Related topics

“Working with server redundancy” on page 137

Referencing objects

This topic describes the rules and conventions applicable to referencing objects, properties and methods.

Referencing the 'attached' object

A script attached to a particular object is considered to be 'within the scope' of that object. This means the script can access the object's properties and methods directly without having to specify the object's name.

For example, in a script attached to a Param object, you could set its Value property as follows:

```
value = "STOP"
```

Referencing the server object

The Server object can be referenced directly in any script regardless of the object to which the script is attached. The following example shows how any script would obtain the value of the StartOfDay property.

```
timeVariable = Server.StartOfDay
```

Referencing other objects

If a script is not attached to the object you want to reference, and the object is part of a collection, you must reference it via its collection.

This example creates a 3600 second (one hour) timer for the point, 'Pump7'.

```
Points("Pump7").CreateTimer 3,3600,0
```

Using methods belonging to other objects

If a script includes a method that is not attached to the object it operates on, you must prefix the method with the object's name.

This example shows how to request a server task (LRN 111) for a point called 'Pump5'.

```
Sub Pump5_OnAcknowledge()  
Server.RequestTask 111, PrmB1k  
End Sub
```

Referencing deleted or non-existent objects

The objects in the Server Scripting Object model are abstractions that represent particular entities in your system—such as points and reports. For example, scripts can create point objects that represent existing points in your system.

However, if you delete an entity from your system, the operation of any scripts that reference that entity will be undefined.

Specifying times and dates

The following formats are commonly used to specify times and dates.

Notes

- VBScript provides many ways of specifying dates and days, such as *vbMonday*. For details, see the help for VBScript.
- Dates are localized for your region.

Full date/time

#mm/dd/yyyy hh:mm:ss#

The exact format is region-specific.

Time (hours, minutes and seconds)

#hh:mm:ss#

If you are using 12hour format and time is after noon, specify 'PM':

*#hh:mm:ss
PM#*

Time (minutes and seconds)

#mm:ss#

Related topics

“Timers” on page 15

Storing script-derived data

Server scripting does not support *global variables* (used to store script-derived data).

However, you can achieve a similar effect to global variables with *user-defined* point parameters. (You first create user-defined parameters for a point, and then use them to store script-derived data.)

The way in which you create user-defined parameters, and the restrictions that apply, depend on the point type.

Storing values with standard points

If you want to store script-derived data with a standard point, you would typically create one or more user-defined parameters for that point. For details about creating user-defined parameters, see Quick Builder's help.

Notes

- If the point is an analog point, you can use its auxiliary parameters—*A1* to *A4*—which are intended for such user-defined tasks.
- You cannot create user-defined parameters for a container point. However, you can create user-defined parameters for its contained (child) points.
- You cannot write to the PV of a point while it is 'on scan'—that is, while it is being scanned by the server. (This restriction applies because the PV parameter is primarily intended to reflect the value of a field device.)

Example scenarios

“Counting the number of pump starts” on page 116

Storing values with flexible points

The technique used to store script-derived data with a flexible point depends on the capabilities of the system interface/point server. Some system interfaces/point servers allow you to create user-defined parameters to store custom data. Alternatively, you can either:

- Use an existing parameter (if a suitable one is available and not being used for other purposes)
- Create a standard point in Quick Builder and use it to store data.

For details, see the documentation for the specific system interface/point server.

Storing values with process points

If you want to store script-derived data with a process point, you can either:

- Use an existing parameter (if a suitable one is available and not being used for other purposes)
- Create a standard point in Quick Builder and use it to store data.

Creating and raising alarms

An alarm that is created in and raised by a script is treated in the same way as any other alarm. For example, if it has a priority other than *Journal*, it appears in the Alarm Summary display and must be acknowledged by an operator.



Attention

- The initial status of the alarm is 'unacknowledged and returned to normal'—as indicated by the use of inverse color for the alarm icon on the Alarm Summary display.

The basic sequence for creating and raising an alarm is:

- 1 Create an AlarmDetails object with the CreateAlarmDetails method using the following syntax:

```
Set AlarmName = Server.CreateAlarmDetails
```

- 2 Define the alarm object's properties, such as its description, priority and so on.
- 3 Raise (generate) the alarm with the GenerateAlarm method.

Example

This example creates an AlarmDetails object called 'AlmDtls', defines its properties, and then raises (generates) the alarm.

```
Sub Tank5_OnTimer()
Dim AlmDtls
'Check whether tank level is too low.
If ... Then
'Level is low, so create, define and generate alarm.
Set AlmDtls = Server.CreateAlarmDetails
AlmDtls.Name = "Tank5"
AlmDtls.Description = "Tank 5 level is too low"
AlmDtls.Priority = hscHigh
AlmDtls.Area = "Basement"
AlmDtls.Value = 20
AlmDtls.Units = "litres"
Server.GenerateAlarm AlmDtls
End If
End Sub
```

Related topics

“AlarmDetails object” on page 70

“CreateAlarmDetails method” on page 86

Comparing strings

If, when comparing strings, you do not know the capitalization of a server-derived string, you should use the VBScript **StrComp** function. For example:

```
If StrComp(EventInfo.type, "CHANGE", vbTextCompare) = 0
then
'Code to run if the event type is CHANGE
End If
```

Using enumerated point parameters

An *enumerated* point parameter is one for which a user-friendly description has been defined for each state of the parameter. For example, the PV of a two-state status point might be enumerated as follows:

State	Enumerated value
0	<i>Stopped</i>
1	<i>Running</i>

ParamValue property

If a parameter's values have been enumerated, the default behavior of the **ParamValue** property is to return/set the enumerated value rather than the state. In the following example *enumvalue* is set to the enumerated value.

```
enumvalue = ParamValue("FIC1234.PV")
```

If you want to return or set the state of an enumerated parameter, you must specify the */ORDN* option. In the following example *state* is set to the state.

```
state = ParamValue("FIC1234.PV/ORDN")
```

Value and ValueOrdn properties

If a parameter's values have been enumerated, the:

- **Value** property returns/sets the enumerated value
- **ValueOrdn** property returns/sets the state

Using server scripting error handlers

You should use error handlers in your scripts so that they operate in an appropriate manner if there is a runtime error. (If error handlers fail to manage a runtime error, the script engine will disable the script.)

For details about error handling, see the *VBScript Reference* from Microsoft.

For a description of error messages specific to server scripting, see the topic titled "*Server scripting error messages*."

Example scenarios

"Changing a report's frequency" on page 109

Related topics

"Server scripting error messages" on page 50

Avoiding operator interaction

Do not write a server script that requires operator interaction because server scripts run in a desktop that is not visible to users.

Similarly, if you use a script to launch an external application, make sure that the application does not pause and wait for input from a user—for example by launching a dialog in Excel or Word (such as save dialog) or by calling MsgBox.

The following examples illustrate **inappropriate** uses for server scripts:

- A script that uses VBScript's MsgBox function
- A script that accesses the Microsoft Outlook Object Model (Outlook will attempt to alert the user that a script is addressing it. If this happens, the message box will never become visible and the script will time out.)
- A script that uses Microsoft Excel in such a way that it expects the user to be able to see a generated graph (However, a script that uses Excel to print the graph would be acceptable.)

If you want to write a script that requires operator interaction, you must write a *display script* (a display-based script). For HMIWeb displays, see HMIWeb Display Builder's help; for DSP displays, see Display Builder's help.

Related topics

“Configuring and managing automatic script engines” on page 62

“Configuring and managing manual script engines” on page 64

“Script engine time-outs” on page 58

“Script object” on page 82

“Launching an external application” on page 123

Launching and interacting with external applications

A server script can launch an external application such as Microsoft Excel or Word, and can then assess it programmatically using its API/object model. For example, you could launch Excel, create a spreadsheet, populate it with appropriate data from your system and then print it.

Notes

- The application cannot be used by (and is not visible to) users. This design limitation has been imposed because there is no guarantee that a user will be present when the script runs and launches the application.
- Make sure that the application does not pause and wait for input from a user.

Example scenarios

“Launching an external application” on page 123

Server redundancy issues

Scripting does not provide direct support for server redundancy. However, redundancy is supported in indirect ways. For example, when the value of a point parameter is changed through the server, it is transferred to the backup server because of the standard redundancy support for point parameters.

If a value is written back into the object model (by using, for example, the **ParamValue** property), it is not immediately saved to disk because it is saved to the memory-resident portion of the database.

This memory-resident portion of the database is written to disk and the backup server every 60 seconds—the process is called *checkpointing*. It may, therefore, take up to 60 seconds for the changes to be committed to disk (which ensures that the changes are permanent). During this period a fault may occur on the server, which could cause the changes to be lost.

This delay is not a problem if the parameter has a *field address* because the change is written to that address at the time of the call, not at the time of the checkpoint. (A field address can either be an address in a controller or an address in a user file.) Consequently, if there is a server failure, the change can be recovered from the field address during the server's initialization phase at startup.

However, if the parameter does not have a field address and the checkpointing delay is unacceptable, the parameter can be immediately committed to disk (and the backup server) with the **CommitValue** method.

Notes

- The **CommitValue** should be used sparingly because the resultant increase in network traffic may degrade server performance.

Example scenarios

“Working with server redundancy” on page 137

Related topics

“CommitValue method” on page 85

Security considerations for custom applications

An application that access the server database via the automation model can be run as:

- An application with an allocated LRN. This is subject to the same security measures as any other application.
- A utility on the server. This requires physical access to the server.

For details about writing applications and utilities, see the *Application Development Guide*.

Using the Script Editor

The way the Script Editor behaves is determined by several factors, including script type and the event to which a script is attached.

You may need to use several applications to perform some tasks. For example, if you want to create an OnAlarm script for a standard point that calls a library script, you must use Quick Builder to write the OnAlarm script, and Station to write the library script.

The topics in this section describe how to write scripts for flexible points or parameters, libraries, periodic scripts, or servers.

Related topics

- “Writing scripts for process points” on page 32
- “Writing scripts for standard points” on page 33
- “Writing scripts for flexible points” on page 34
- “Writing scripts for alerts” on page 35
- “Writing scripts for the server, library or periodic scripts” on page 36
- “Writing scripts for reports” on page 38
- “Script Editor basics” on page 39

Writing scripts for process points

You use the Script Editor embedded in Control Builder to write scripts for *process points* (Process Controller points).

Scripts are stored as part of the control module definition.

**Attention**

You can view a point's scripts in Station by choosing **Configure > Server Scripting > Point Scripts**. However, if you want to edit the script, you must do it in Control Builder.

To write a script for a point:

- 1 Open Control Builder.
- 2 Right-click the point and choose **Configure Module Parameters**.
- 3 Click the Server History tab.
- 4 Click the **Create New or Edit Existing Server Scripts** button to open the Script Editor.
- 5 Select the event from **Event** and, if appropriate, the parameter from **Parameter**.
- 6 Click inside the editing area and start writing the script.

Related topics

“Script Editor basics” on page 39

“Script Editor basics specific to Station” on page 40

“Testing and debugging scripts” on page 43

“Server Scripting Object Model reference” on page 67

“Example server scripting scenarios” on page 103

Writing scripts for standard points

You use the Script Editor embedded in Quick Builder to write scripts for *standard points* (Experion's inbuilt points).

You can only write a script for one point at a time. However, you can copy a script from one point and paste it into another point.



Tip

You can view a point's scripts in Station by choosing **Configure > Server Scripting > Point Scripts**. However, if you want to edit a point's scripts, you must do that in Quick Builder and then download the point again.

To write a script for a point

- 1 Open the Quick Builder project that defines the point.
If you cannot find the Quick Builder project that contains the original (*draft*) script, you can upload the point's details—which includes its scripts—into Quick Builder, edit the script and then download the point.
- 2 Select the point and click the Script tab.
- 3 Click **Create New or Edit Existing Server Scripts** to open the Script Editor.
- 4 Select the event from **Event** and, if appropriate, the parameter from **Parameter**.
- 5 Click inside the editing area and start writing the script.



Tip

To call up the *VBScript Reference*, choose **Help > VBScript Reference** from Quick Builder's menu.

- 6 Download the point to the server.
The scripts can only be run after the project, or the relevant points, have been downloaded to the server—at which time they become *online*.


Related topics

- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103

Writing scripts for flexible points

You use the Script Editor embedded in Station to write scripts for *flexible* points (points on a system interface or point server).

To write a script for a flexible point:

- 1 Choose **Configure > Server Scripting > Point Scripts** to call up the Server Scripting - Point Scripts display.
- 2 Click  to open the Point Browser.
- 3 Search for and select the point and then click **Apply** to insert it in the Script Editor's point field.
- 4 If the Script Editor's Draft tab is not at the front, click it to bring it to the front.
- 5 Select the event from **Event** and, if appropriate, the parameter from **Parameter**.
- 6 Click inside the editing area and start writing the script.
- 7 When you have finished, click **Commit Draft to Online**.

Related topics

- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103

Writing scripts for alerts

You use the Script Editor embedded in Station to write scripts for *alerts*. Alerts are notifications whose urgency and priority are not high enough to be alarms.

An alert has a *source* (the object for which the alert is generated) and a *condition* (the reason that caused the alert to be generated).

You can write a single script for a source, which applies to all possible conditions associated with the source. Alternatively, you can write several scripts, each of which applies to a specific condition associated with the source.

To write a script for an alert

- 1 Type **syscfgscriptingalert** in the Command Zone and press ENTER.
- 2 In the **New Alert Source** box, type the name of the source.
- 3 If appropriate, in the **New Condition** box, type the name of the condition.
- 4 If the Script Editor's **Draft** tab is not at the front, click it to bring it to the front.
- 5 In the **Event** list, click the event.
- 6 Click inside the editing area and start writing the script.
- 7 When you have finished, click **Commit Draft to Online**.

To edit an existing script for an alert

- 1 Type **syscfgscriptingalert** in the Command Zone and press ENTER.
- 2 Click **Edit Existing Scripts**.
- 3 In the **Existing Alert Source** list, click the source.
- 4 If appropriate, in the **Existing Condition** list, click the condition.
- 5 If the Script Editor's **Draft** tab is not at the front, click it to bring it to the front.
- 6 In the **Event** list, click the event.
- 7 Edit the script as required.
- 8 When you have finished, click **Commit Draft to Online**.

Related topics

- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103

Writing scripts for the server, library or periodic scripts

You use the Script Editor embedded in Station to write scripts for the server, library or periodic scripts.

To write a script for the server, library or periodic script:

- 1 Click **Server Scripts** on the System Configuration display to call up the Server Scripting display.
- 2 If the Script Editor's Draft tab is not at the front, click it to bring it to the front.
- 3 Select the type of script you want to edit from **Script Type**.
- 4 If you selected:
 - *server*, select the **Event**, and then click inside the editing area and start writing the script.
 - *periodic*, click **Add**, and then click inside the editing area and start writing the script.
 - *library*, click at the bottom of the list of existing scripts inside the editing area and start writing the script.

(Unlike other types of scripts, the library scripts appear on the same 'page' of the Script Editor. However, after you have created a library script, you can go directly to it by selecting it from the **Procedure** list.)

- 5 If the script is a periodic script, configure its properties.
- 6 When you have finished, click **Commit Draft to Online**.



Attention

In the case of a library script, at least one non-library script must already be online before you can commit the library script to online.

In the case of a periodic script, because a periodic script is not associated with any particular object, you must fully qualify any object referenced in the script. For example, when attempting to read or write a parameter value of point, fully qualify the point as a child of the Server object: 'Server.ParamValue'.

Related topics

- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103

Periodic script properties

This topic describes the properties for periodic scripts.

Property	Description
Name	The script's name. 25 characters maximum.
Period	The frequency at which the script runs.
Units	The minimum period is 1 second. To minimize the load on the system, you should make the period as long as possible.
Start Time	The date/time at which the script runs for the first time. If no date/time is specified, the script runs for the first time when the scripting subsystem starts.

Property	Description
End Time	The date/time at which the script no longer runs. If no date/time is specified, the script runs at the specified period until the scripting subsystem stops.
Script Engine	The script engine that runs the script.

Writing scripts for reports

You use the Script Editor embedded in Station to write scripts for reports.

To write a script for a report:

- 1 Click **Reports** on the System Menu display to call up the Reports display.
- 2 Click the report for which you want to write a script to see the report's basic configuration details.
- 3 Click **Configure** to see the report's configuration details.
- 4 Click the Scripting tab.
- 5 If the Script Editor's Draft tab is not at the front, click it to bring it to the front.
- 6 Select the **Event**.
- 7 Click inside the editing area and start writing the script.
- 8 When you have finished, click **Commit Draft to Online**.

Script Editor basics

This topic is applicable to the Script Editor, irrespective of the application through which you access it.

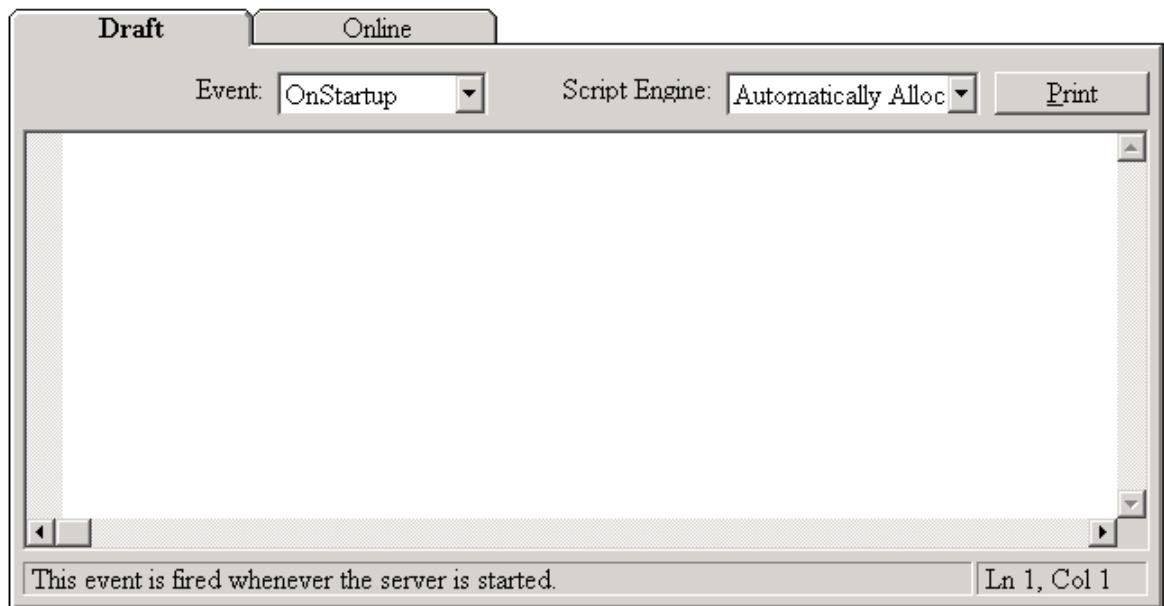


Figure 2: The Script Editor (as it appears in Quick Builder)

Notes

- Scripts are color-coded as you type so that you can easily identify the various components.

Color	Component
 	VBScript grammar
 	Objects, properties, methods and values
 	Comments

- The Script Editor performs basic syntax checking when you press ENTER or move to another line, and highlights syntax errors. (Note that Script Editor cannot detect errors that only become apparent at run time.)
- Events that have scripts for the selected object are highlighted in the **Event** list.
- You can save a lot of time by copying script code from this help, pasting it into your own scripts, and then modifying it accordingly.
- Deleting the contents of a script also deletes the script.

Related topics

- “Script Editor basics specific to Station” on page 40
- “Writing scripts for alerts” on page 35
- “Writing scripts for flexible points” on page 34
- “Writing scripts for process points” on page 32
- “Writing scripts for standard points” on page 33
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103

“Writing scripts for the server, library or periodic scripts” on page 36

“Copying example Script Editor scripts” on page 105

Script Editor basics specific to Station

This topic is only applicable if you are using the Script Editor in Station, and supplements the information in the topic 'Script Editor basics'.

Notes

- When you first create a script, you create a *draft* version. When you have finished editing it, you create an *online* version (which becomes immediately available for execution).
- You must have *ENGR* or *MNGR* security level to write scripts.
- You can view—but not edit—the online script by clicking the Online tab.
- If you call up another display when editing the draft copy of a script, a dialog box appears, asking whether you want to save your changes.
- To call up the *VBScript Reference*, choose **Help > VBScript Reference** from Station's menu.

Using the Script Editor's buttons

To:	Click:
Save changes made to the draft script	Save Draft Changes
Update the online script, so that it is the same as the draft script	Commit Draft to Online
Undo changes made to the draft script since the last save.	Cancel Draft Changes
Undo all changes to the draft script. (Overwrite the draft script with the online script.)	Reset Draft from Online

Related topics

“Script Editor basics” on page 39

“Writing scripts for alerts” on page 35

“Writing scripts for flexible points” on page 34

“Writing scripts for process points” on page 32

“Writing scripts for standard points” on page 33

“Testing and debugging scripts” on page 43

“Server Scripting Object Model reference” on page 67

“Example server scripting scenarios” on page 103

“Writing scripts for the server, library or periodic scripts” on page 36

Script Editor keyboard shortcuts

This topic lists the keyboard shortcuts available in the script editor.

Command	Shortcut
Copy	CTRL+C
Cut	CTRL+X
Find	CTRL+F
Help	F1

Command	Shortcut
Paste	CTRL+V

Testing and debugging scripts

This section describes how to test and debug your scripts.



Attention

Honeywell recommends that you use the predefined 'Testing' manual script engine to test and debug your scripts before reassigning them to other script engines.

To learn about:	Go to:
Forcing a script to run by simulating the associated event	“Simulating events” on page 44
Script debugging	“Script debugging” on page 45
Techniques for checking script progress	“Checking script progress” on page 48
Interpreting script-related entries in the server log	“Interpreting the server log” on page 49
Server scripting error messages	“Server scripting error messages” on page 50

Related topics

- “Simulating events” on page 44
- “Script debugging” on page 45
- “Checking script progress” on page 48
- “Interpreting the server log” on page 49
- “Server scripting error messages” on page 50
- “Server Scripting Object Model reference” on page 67
- “Example server scripting scenarios” on page 103
- “Writing scripts for alerts” on page 35
- “Writing scripts for flexible points” on page 34
- “Writing scripts for process points” on page 32
- “Writing scripts for standard points” on page 33
- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Writing scripts for the server, library or periodic scripts” on page 36

Simulating events

When testing a script, you generally need to simulate an event in order to force the script to run—for example, it is not feasible to reboot the server simply to test an OnStartup script.

You simulate events using the FireScriptEvent method and the EventInfo object.

You have written a script for the OnAlarm event for a point called 'Tank1', and want to test it without putting the point into alarm.

You write a 'test' script that simulates the OnAlarm event for Tank1. This script:

1. Defines the simulated alarm's details using the EventInfo object
2. Raises the simulated alarm using the FireScriptEvent method

The test script is a library script, so that you can easily change it if you want to perform more tests. (If you attach the script to a point, you would have to download the point from Quick Builder each time you changed the script.)

In order to run the test script (library scripts only run if called by another script), you configure a status point called 'TestPoint' to monitor the status of a switch—pressing the switch runs the point's OnChange script which, in turn, runs the test script.

The test script:

```
Sub TestScript()
'Define the simulated alarm
EventInfo.Area = "A1"
EventInfo.Description = "Tank is full"

'Now raise the simulated the alarm
Server.Points("Tank1").FireScriptEvent hscPointEventOnAlarm
End Sub
```

The OnChange script for TestPoint:

```
Sub TestPoint_PV_OnChange()
'Call the test (library) script
TestScript
End Sub
```

Related topics

“FireScriptEvent method” on page 88

“EventInfo object” on page 74

Script debugging

Use Microsoft Visual Studio to test and troubleshoot your scripts. Using Visual Studio, you can:

- View the source code of the script you are debugging.
- Control the pace of script execution with break points and stepping.
- View and change *variable* and *property* values with the **Immediate** window and **Watch** window.
- View and control script flow with the **Call Stack** window.

If you are debugging Server scripts, the Visual Studio must be installed on the Server. For Station scripting, Visual Studio must be installed on the Station computer.

Script debugging runs under the Windows *mng*r account (configured for Experion). You must have rights to use this account and be signed on using this account before you can debug scripts.

Configuring Internet Explorer for debugging Station scripts

To debug Station scripts, Microsoft Internet Explorer (IE) on the Station computer must be configured to enable script debugging.



Attention

This step is only required for debugging Station scripts. If you want to debug Server scripts, you need only follow 'Configuring Microsoft Visual Studio to debug server scripts'.

To configure Internet Explorer to debug Station scripts

- 1 Log on using the Windows User *mng*r account.
- 2 Start Internet Explorer and check whether the **Internet Connection Wizard** appears:
 - If the wizard does not appear, go to step 4. (This means that Internet Explorer has already been configured under the *mng*r account.)
 - If the wizard appears, continue with step 3.
- 3 Use the wizard to configure Internet Explorer as follows:
 - a Click **I want to setup my Internet connection manually...**, and then click **Next**.
 - b Click **I connect through a Local Area Network (LAN)**, and then click **Next**.
 - c Click **Next**.
 - d Click **No**, and then click **Next**.
 - e Click **Finish**.
If you do not have an Internet connection, you will receive an error message. You can safely ignore this message.
- 4 Select **Tools > Internet Options > Advanced** tab.
- 5 Clear the **Disable Script Debugging (Other)** check box.
- 6 Click **OK**.
- 7 Restart Station for the change to take effect.

Configuring Microsoft Visual Studio to debug server scripts

To debug server scripts, you need to install and configure Microsoft Visual Studio.



Tip

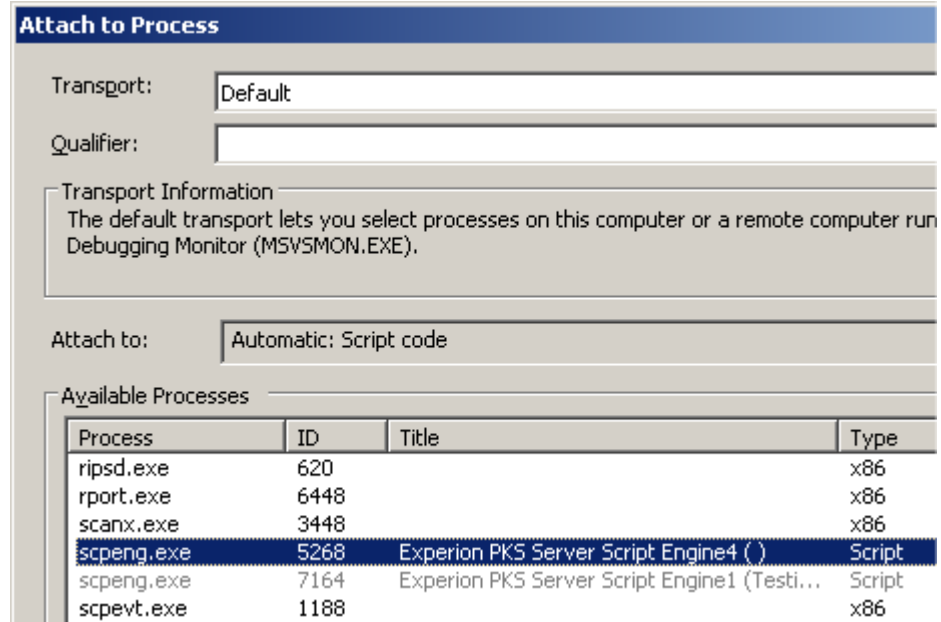
If you want to debug Station scripts, you need only follow 'Configuring Internet Explorer for debugging Station scripts'.

Prerequisites

- You have installed Microsoft Visual Studio on the server.

To configure Microsoft Visual Studio to debug server scripts

- 1 Log on to the Windows operating system using the Windows *mng*r account.
- 2 Start Microsoft Visual Studio, and choose **Tools > Attach To Process**.
- 3 Click **scpeng.exe**.



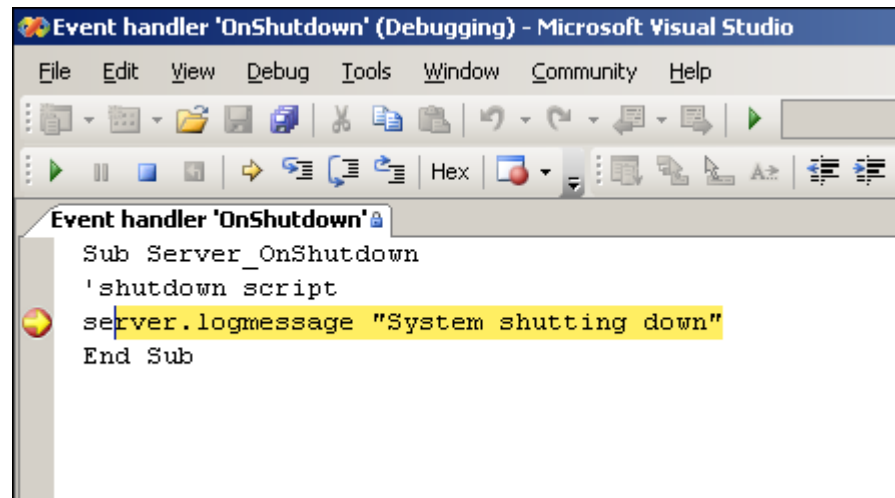
There may be multiple entries of *scpeng.exe*, one for each running Script engine on the Server. Examine the **Title** to determine the name of the Script engine. For example **Experion PKS Server Script Engine Automatic** for an automatic script engine or **Experion PKS Server Script Engine1** for manual engine 1.

- 4 Click **Attach**. If a security warning is shown, click **Attach**.
- 5 Select the **Script Explorer** window to view the scripts that are available for debugging.

**Tip**

If the **Script Explorer** window cannot be seen, choose **Debug > Windows > Script Explorer**.

- 6 Select a script to debug. You can insert breakpoints into the script so that when the script runs (when the script event is fired) the debugger will catch the event and pause the script at the breakpoint for you to debug. For further debugging instructions, see the debugger's help.



Checking script progress

Server scripting does not support Message Boxes. Consequently, you cannot use them to show a script's progress.

However, you can include a simple 'error handler' in a script that writes suitable progress comments to a point parameter. You can then create a suitable display that shows the last comment to be successfully written to the parameter. (The display's update rate is far too slow to show progress in real time.)

You include the following error handler at appropriate points in your script. This writes the step number, *nn*, to a user-defined point parameter called 'Debug'. (For details about creating user-defined parameters, see the *Quick Builder User's Guide*.)

```
ParamValue(".Debug")= nn
```

A typical script would then look like this:

```
dim period
period=ParamValue("TankFarmStatus.IncrementPeriod")
ParamValue(".Debug")= 10
'Check if drain valve is closed
if ParamValue(".PV")="CLOSED" then
    ParamValue(".Debug")= 20
    'Decrement level by 5 via timer
    Point.CreateTimer 1,period,0
    ParamValue(".Debug")= 30
else
    ParamValue(".Debug")= 40
    Point.KillTimer 1
    ParamValue(".Debug")= 50
end if
ParamValue(".Debug")= 60
```

Interpreting the server log

The server log reports the following types of run-time problems with server scripts:

- Syntax errors (as shown in the following example)
- Attempts to perform an illegal action, such as an attempt to read the value of a parameter that does not exist.

To see the server log, choose **Start > All Programs > Honeywell Experion PKS > Server > Diagnostic Tools > Experion PKS Server Log**.

The following figure shows a typical entry for a script-related error. The entry specifies:

- The event (*onTimer*)
- The object (*Timer HourlyUpdate*). Because this example relates to a periodic timer, the name is of the form *Timer TimerName*.
- The script engine that was running the script (*1*)

Note that the script engine will be identified by its name if it has one; otherwise, it will be identified by its manual/automatic engine number.

- The error's location in the script (*line 3 at position 3*)

```
27-Nov-01 07:49:10.185 (3196 3220) CScriptSite.cpp,v:2706: Script error on eve
nt >onTimer< for object >Timer HourlyUpdate< running in Script Engine 1
27-Nov-01 07:49:10.185 (3196 3220) CScriptSite.cpp,v:2707: Error 0x800a01b6:>O
bjeet doesn't support this property or method: 'Server.CreateAlarmDetail'<
27-Nov-01 07:49:10.185 (3196 3220) CScriptSite.cpp,v:2708: Error occurred on li
ne 3 at position 3
27-Nov-01 07:49:10.195 (1740 3208) cmanageevent.cpp,v:224: Error received for
engine 1
```

Related topics

“Server scripting error messages” on page 50

Server scripting error messages

The following run-time error messages in the server log are specific to server scripting.

Related topics

“0x80004005 (E_FAIL)” on page 50
 “0x80040801 (0x80040000 | M4_BUSY_TRB)” on page 50
 “0x80040802 (0x80040000 | M4_ILLEGAL_LRN)” on page 51
 “0x80041001 (E_ITF_IN_COLLECTION)” on page 51
 “0x8004101E (E_ITF_OBJ_NOT_CONFIGURED)” on page 51
 “0x80041002 (E_ITF_NOT_IN_COLLECTION)” on page 51
 “0x80048212 (0x80040000 | M4_PNT_ON_SCAN)” on page 51
 “0x80048231 (0x80040000 | M4_INV_POINT)” on page 52
 “0x80048232 (0x80040000 | M4_INV_PARAMETER)” on page 52
 “0x800483B0” on page 52
 “0x800482B2 (0x80040000 | M4_PARAM_NAME_EXISTS)” on page 52
 “0x800482B4 (0x80040000 | M4_INV_AREA)” on page 53
 “0x800483FA” on page 53
 “0x80070005 (E_ACCESSDENIED)” on page 53
 “0x80070057 (E_INVALIDARG)” on page 53
 “Using server scripting error handlers” on page 25
 “Interpreting the server log” on page 49

0x80004005 (E_FAIL)

The script failed.

Cause

The reason for the failure error depends on the type of object:

- **Alarm.** There was an attempt to generate an alarm on a remote location (that is, a location not associated with the server on which the script ran).
- **File.** A problem occurred when sending the file to the backup server in a redundant server system.
- **Report.** There was a request for a non-requestable report.

Solution

In the case of:

- **Alarm.** Move the script to the server associated with the location.
- **File.** Determine why the file cannot be sent to the backup server.
- **Report.** Reconfigure the report so that it is requestable.

0x80040801 (0x80040000 | M4_BUSY_TRB)

The requested task was busy.

Cause

The script attempted to call a task that was busy.

Solution

Add logic to the script that checks whether the task is busy before requesting it.

0x80040802 (0x80040000 | M4_ILLEGAL_LRN)

The requested task does not exist.

Cause

The script attempted to call a task that does not exist.

Solution

Correct the script.

0x80041001 (E_ITF_IN_COLLECTION)

The specified timer already exists.

Cause

The script attempted to create a timer that already exists.

Solution

Correct the script.

0x8004101E (E_ITF_OBJ_NOT_CONFIGURED)

The specified asset does not exist.

Cause

The script specified an asset in the AlarmDetails object that does not exist.

Solution

Correct the script.

0x80041002 (E_ITF_NOT_IN_COLLECTION)

The specified timer does not exist.

Cause

The script attempted to modify or kill a non-existent timer.

Solution

Correct the script.

0x80048212 (0x80040000 | M4_PNT_ON_SCAN)

The specified point is on scan.

Cause

The script attempted to write to a point's PV while the point was on scan. (You can only write to the PV if the point is off scan.)

Solution

Modify the script so that it does not need to write to the PV.

0x80048231 (0x80040000 | M4_INV_POINT)

The requested point cannot be found.

Cause

The script referenced a point that could not be found because it:

- Does not exist.
- Is it remote and temporarily unavailable.

Solution

Check whether the point exists, or why it was unavailable.

Otherwise, correct the script.

0x80048232 (0x80040000 | M4_INV_PARAMETER)

The specified parameter does not exist.

Cause

The script referenced a parameter that does not exist.

Solution

Correct the script.

0x800483B0

The DLL server instance already exists.

Cause

The script attempted to create more than one instance of the DLL server. (The DLL server can be initialized as a free-threaded or an apartment-threaded server—depending upon the CoInitialize call within the client application. If it is started as a free-threaded server, multiple other free-threaded instances of it can be created; otherwise, there can only be one instance of it.)

Solution

Correct the script.

0x800482B2 (0x80040000 | M4_PARAM_NAME_EXISTS)

The specified parameter already exists.

Cause

The script attempted to create a parameter that already exists.

Solution

Correct the script.

0x800482B4 (0x80040000 | M4_INV_AREA)

The specified assignable asset does not exist.

Cause

The script referenced an assignable asset that does not exist.

Solution

Correct the script, or define the assignable asset.

0x800483FA

The server database is not loaded.

Cause

The script attempted to create a server object without the database being loaded.

Solution

Ensure that the script does not run if the database is not loaded.

0x80070005 (E_ACCESSDENIED)

The specified parameter is read-only.

Cause

The script attempted to set a read-only parameter.

Solution

Correct the script.

0x80070057 (E_INVALIDARG)

An argument is invalid.

Cause

The script passed an invalid argument. The meaning depends on the type of object:

- *File*. The specified file does not exist.
- *Program*. The specified program does not exist.
- *Report*. The report with the specified number was not found.

Solution

Correct the script.

Script engines

A *script engine* is a software module that interprets a script and performs its instructions.

To learn about:	Go to:
The way in which scripts are assigned to script engines	“Assigning scripts to script engines” on page 56
Script queues	“Script queues” on page 57
Script timeouts	“Script engine time-outs” on page 58
The operation of script engines when the server starts up and shuts down	“Server startup and shutdown” on page 59

Related topics

- “Assigning scripts to script engines” on page 56
- “Script queues” on page 57
- “Script engine time-outs” on page 58
- “Server startup and shutdown” on page 59

Assigning scripts to script engines

When you first write a script, the server initially assigns it to an *automatic* script engine in a manner that optimizes scripting performance. The number of automatic script engines you need depends on the extent to which you use scripts. If you only have a few scripts, you probably only need to enable one automatic script engine. However, as you add more scripts, you should monitor scripting performance and enable more automatic script engines as appropriate. You cannot assign scripts to a specific automatic script engine.

However, you should consider reassigning a script to a manual script engine in the following cases:

- You are debugging the script. We recommend that you use the predefined 'Testing' manual script engine to test and debug your scripts before reassigning them to other script engines (automatic or manual).
- The script takes a long time to run. For example, you may want to assign long-running report scripts to a particular script engine, so that they do not interfere with more urgent server and point scripts.
- The script is related to other scripts, and you want them to run in a particular order. Because there is no synchronization between script engines, related scripts must be assigned to the same script engine if you want them sequentially. (Note that, even when assigned to the same script engine, the order in which scripts are run is determined by the order in which the associated events occur.)
- You want to be able to simultaneously disable/enable a set of related scripts—when you disable a manual script engine, you also disable all its scripts.

It is a good idea to give each manual script engine an appropriate name before assigning scripts to it.

Related topics

“Configuring and managing script engines” on page 61

Script queues

A script engine will queue scripts if their events occur while the script engine is running an earlier script. The script engine then runs the queued scripts in the order in which their events occurred.

Each script engine maintains its own queue, and there is no synchronization between queues. This means that if you want to run scripts in a particular order, you must assign them to the same script engine. (However, even when assigned to the same script engine, the order in which scripts are run is determined by the order in which their events occur.)

Script engine time-outs

The script engine *time-out* is the maximum time a script engine has to run a script before terminating it.

The following table describes what happens if a script times out.

Reason for time-out	Result
An infinite or unreasonably long loop	The script engine: <ul style="list-style-type: none"> • Disables the script • Raises an alarm
A function call times out	The script engine: <ul style="list-style-type: none"> • Disables the script • Raises an alarm • Restarts (Events with scripts for the script engine are queued during the restart—but if the queue fills up, any subsequent events are discarded.)
A library function times out	The script engine: <ul style="list-style-type: none"> • Disables the script • Raises an alarm against the calling script, not the function

Notes

- There is one time-out for all scripts assigned to automatic script engines. However, each manual script engine can have a separate time-out.
- In general, you should use the default time-out period. If you set the time-out to a large value and a script (for whatever reason) stops running, any other pending scripts for that script engine will be prevented from running until the time-out expires.
- You can use the `script` object to change the time-out for the current script. However, the time-out returns to the value specified for the script engine as soon as the next script runs.
- A script will time out if it requests user input.

Related topics

“Script object” on page 82

“Configuring and managing automatic script engines” on page 62

“Configuring and managing manual script engines” on page 64

“Avoiding operator interaction” on page 26

“Reenabling scripts assigned to manual script engines” on page 65

Server startup and shutdown

OnStartup event

When the server is started, all currently enabled script engines are started (if they have scripts assigned to them), and the OnStartup script is run.

OnShutdown event

When the server is shut down:

- Events queued to the script engines prior to the shutdown request are discarded.
- If no scripts are running, the OnShutdown script is run. (It will be terminated if it does not complete before timeout period.)
- If a script engine is running a script, it waits for its timeout period before terminating the script, and then running the OnShutdown event.



Attention

The automation model is unavailable for use within the OnShutdown event. For example, the script cannot use any methods or access an object's properties.

Custom applications that use the Server Scripting Object Model

If you have custom applications that interact with the Server Scripting Object Model, you can configure them to start when the server starts. For details, see the topic '*Starting a task automatically*' in the *Application Development Guide*.

When the server shuts down these applications are sent a shutdown request.

Note that custom applications that use the Server Scripting Object Model to interact with the server won't respond to task requests.

Configuring and managing script engines

You use **Station** to configure and manage script engines.



Attention

- You need *ENGR* or *MNGR* security level to configure and manage script engines.

To learn how to:	Go to:
Configure and manage automatic script engines	“Configuring and managing automatic script engines” on page 62
Re-enable scripts assigned to automatic script engines	“Reenabling scripts assigned to automatic script engines” on page 63
Configure and manage manual script engines	“Configuring and managing manual script engines” on page 64
Re-enable scripts assigned to manual script engines	“Reenabling scripts assigned to manual script engines” on page 65
Configuring debugging options	“Configuring debugging options” on page 66

Related topics

“Configuring and managing automatic script engines” on page 62

“Configuring and managing manual script engines” on page 64

“Configuring debugging options” on page 66

“Assigning scripts to script engines” on page 56

“Setting a point to a specific value at the start of each day” on page 133

Configuring and managing automatic script engines

The number of automatic script engines you need depends on the extent to which you use scripts. If you only have a few scripts, you probably only need to enable one automatic script engine. However, as you add more scripts, you should monitor scripting performance and enable more automatic script engines as appropriate.

If no automatic script engine is enabled, then no scripts assigned to automatic script engines will run.



Attention

- Clicking **Reenable Scripts** re-enables scripts assigned to automatic script engines that have timed out or have syntax errors.

To configure/manage automatic script engines:

- 1 Click **Script Engines** on the **System Configuration** display to call up the **Script Engines** display.
- 2 Click the **Automatic Engines** tab.
- 3 Configure the script engines as required.

Related topics

“Configuring and managing manual script engines” on page 64


“Avoiding operator interaction” on page 26

“Script engine time-outs” on page 58

“Script object” on page 82

Automatic script engine properties

Property	Description
Enabled	Select this check box to enable the script engines.
Status	Indicates the script engine's current status, which can be: <ul style="list-style-type: none"> • <i>Not running</i> • <i>Running</i> • <i>Failed script/s</i> • <i>Failed</i>
Total scripts	The number of scripts assigned to the script engine.
Scripts with syntax errors	The number of scripts that contain syntax errors.
Scripts with run-time errors	The number of scripts that have run-time errors and have been disabled.
Timed out scripts	The number of scripts that have timed-out and have been disabled.
Event rate	The number of events that were processed by the script engine in the last minute.
Configuration	
Number of engines in use	Specifies the maximum number of automatic engines that can run simultaneously. Changing this may cause scripts to be redistributed among script engines.
Script execution timeout	The timeout period for all scripts assigned to automatic script engines.

Property	Description
Journal parameter changes	<p>Select this check box if you want to journal to the Event log changes in parameter values that are caused by scripts.</p> <p>The journal only indicates the object and event that made the change, which might not be the object and event that were initially called.</p> <hr/> <p> Attention</p> <p>Journaling parameter value changes can place a significant load on the server. An alternative is to use the GenerateAlarm method to journal only significant changes.</p> <hr/>
Log script events	Select this check box if you want to log script-related events to the Server log.

Reenabling scripts assigned to automatic script engines

If a script times out or has syntax errors, you need to re-enable it (after fixing the problem).

To re-enable disabled scripts:

- 1 Click **Script Engines** on the **System Configuration** display to call up the **Script Engines** display.
- 2 Click the **Automatic Engines** tab.
- 3 Click **Reenable Scripts**.

Configuring and managing manual script engines

You need to individually configure each manual script engine that you want to use.



Attention

Clicking **Reenable Scripts** re-enables scripts for the selected script engine that have timed out or have syntax errors.

To configure a manual script engine:

- 1 Click **Script Engines** on the **System Configuration** display to call up the **Script Engines** display.
- 2 Click the **Manual Engines** tab.
- 3 Click the script engine's name to call up the configuration display. (If you have not yet defined a name, click the hyphen that appears in the **Name** column.)
- 4 Configure its properties as appropriate.



Tip

To configure another manual script engine, select it from the **Script Engine** list at the top of the display.

Related topics

“Configuring and managing automatic script engines” on page 62


“Avoiding operator interaction” on page 26

“Script engine time-outs” on page 58

“Script object” on page 82

Manual script engine properties

Property	Description
Name	The name of the currently selected script engine.
Enabled	Select this check box to enable the script engine.
Status	Indicates the script engine's current status, which can be: <ul style="list-style-type: none"> • <i>Not running</i> • <i>Running</i> • <i>Failed script/s</i> • <i>Failed</i>
Total Scripts	The number of scripts assigned to the script engine. Click List Scripts to see the list of scripts.
Scripts with Syntax Errors	The number of scripts that contain syntax errors.
Scripts with run-time errors	The number of scripts that have run-time errors and have been disabled.
Timed out scripts	The number of scripts that have timed-out and have been disabled.
Event Rate	The number of events that were processed by the script engine in the last minute.
Configuration	
Script execution timeout	The timeout period for scripts.

Property	Description
Journal parameter changes	<p>Select this check box if you want to journal changes in parameter values that are caused by scripts.</p> <p>The journal only indicates the object and event that made the change, which might not be the object and event that were initially called.</p> <hr/> <p> Attention</p> <p>Journaling parameter value changes can place a significant load on the server. An alternative is to use the GenerateAlarm method to journal only significant changes.</p> <hr/>
Log script events	Select this check box if you want to log script-related events.

Related topics

“Reenabling scripts assigned to manual script engines” on page 65

Reenabling scripts assigned to manual script engines

If a script times out or has syntax errors, you need to re-enable it (after fixing the problem).

To re-enable disabled scripts:

- 1 Click **Script Engines** on the System Configuration display to call up the Script Engines display.
- 2 Click the **Manual Engines** tab.
- 3 Click the script engine's name to call up its configuration details.
- 4 Click **Reenable Scripts**.

Related topics

“Manual script engine properties” on page 64

“Script engine time-outs” on page 58

Configuring debugging options

You can optionally set the script engines to operate in debug mode for debugging of your scripts. Once set (or unset), you must restart the script engines for the change to take effect.

To configure script debugging:

- 1 Click **Script Engines** on the **System Configuration** display to call up the Script Engines display.
- 2 Click the **Debugging Options** tab.
- 3 Select or clear the **Run scripts in debug mode** check box as appropriate.
If selected, causes the script engines to operate in debug mode at their next restart.
- 4 You must restart the script engines for the setting to take effect.

Server Scripting Object Model reference

This section provides an overview of the server scripting object model, as well as descriptions of each object, method and event.

Related topics

- “Overview of the Server Scripting Object Model” on page 68
- “Objects” on page 70
- “Methods” on page 85
- “Events” on page 96
- “Testing and debugging scripts” on page 43
- “Example server scripting scenarios” on page 103
- “Writing scripts for alerts” on page 35
- “Writing scripts for flexible points” on page 34
- “Writing scripts for process points” on page 32
- “Writing scripts for standard points” on page 33
- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Writing scripts for the server, library or periodic scripts” on page 36

Overview of the Server Scripting Object Model

The *Server Scripting Object Model* represents each important part or aspect of your system as an *object*—for example, there is a point object and a report object. When writing scripts, you treat these objects almost as if they were real objects—for example, if you wanted to generate a report, your script would first 'select' the report and then issue a 'generate' instruction.

You should also familiarize yourself with the following terms, which are used extensively in this document:

- **Collection.** A high-level object that represents all objects of the same type. For example, the Points collection represents all Point objects.
- **Properties.** The object's attributes or characteristics. The AlarmDetails object, for example, has properties such as Priority and Area. Most properties can be written to, but some are read-only.
- **Methods.** The 'functions' or 'commands' that are applicable to the object. The Report object, for example, has a Request method that requests the report.
- **Events.** Changes or actions that affect the object in some way. The Point object, for example, has an OnAlarm event which occurs when the point goes into alarm.

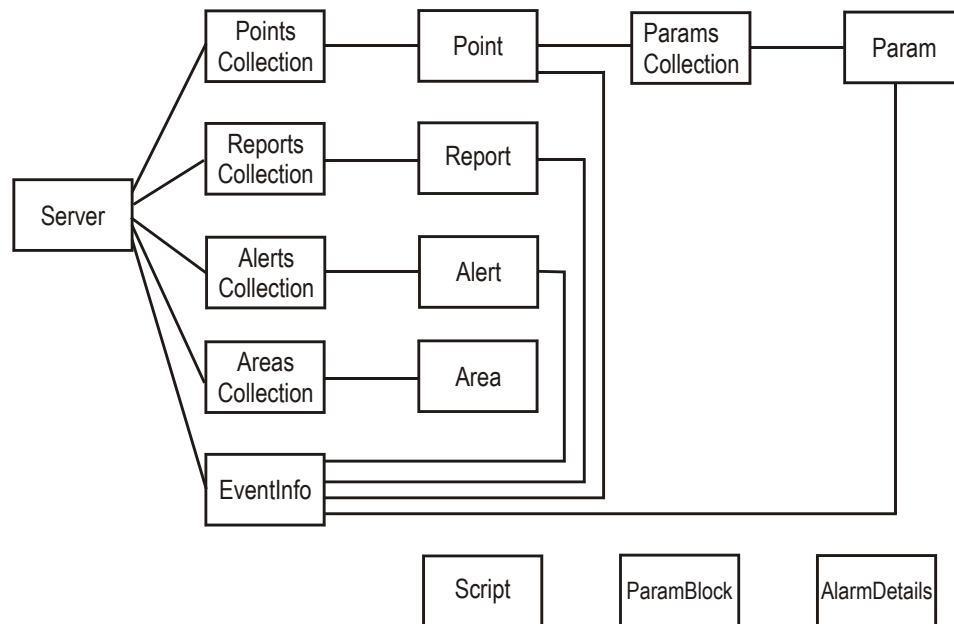


Figure 3: Server Scripting Object Model

Object type	Description
Server object	The Server object is at the top of the object hierarchy, and represents the server application.
Point object	Represents the high-level aspects of a point.
Points collection	Represents all points.
Param object	Represents a point parameter. For example, you use the Param object to obtain or set a parameter's value.
Params collection	Represents all point parameters for a particular point.
Report object	Represents a report.
Reports collection	Represents all reports.
Alert object	Represents an alert.

Object type	Description
Alerts collection	Represents all alerts.
Area object	Represents an asset. You can use the Area object to, for example, find out how many unacknowledged alarms there are for an asset.
Areas collection	Represents all assets.
AlarmDetails object	A temporary storage area that is used by scripts when generating alarms.
EventInfo object	Represents the event that has just occurred.
ParamBlock object	A temporary storage area used by scripts when requesting a task (LRN). The object's contents are passed to the task when the task is requested.
Script object	Represents the currently executing script.

Objects

The objects are listed in alphabetical order.

Related topics

- “AlarmDetails object” on page 70
- “Alert object” on page 71
- “Alerts collection” on page 72
- “Area object” on page 72
- “Areas collection” on page 73
- “EventInfo object” on page 74
- “Param object” on page 77
- “ParamBlock object” on page 78
- “Params collection” on page 79
- “Point object” on page 79
- “Points collection” on page 80
- “Report object” on page 81
- “Reports collection” on page 82
- “Script object” on page 82
- “Server object” on page 83

AlarmDetails object

Description

The AlarmDetails object is used as a temporary storage area when creating an alarm.

Properties

Property	Description
<i>Area</i>	Returns or sets the assignable asset to which the alarm belongs. Appears in the Location column on the Alarm Summary display.
<i>Description</i>	Returns or sets the alarm text. Appears in the Description column on the Alarm Summary display.
<i>Name</i>	The name of the object to which the alarm belongs, such as a point or controller. Appears in the Source column on the Alarm Summary display.
<i>Priority</i>	Returns or sets the alarm's priority: <ul style="list-style-type: none"> • <i>hscUrgent</i> • <i>hscHigh</i> • <i>hscLow</i> • <i>hscJournal</i> (default) Appears as the letter prefix in the Priority column on the Alarm Summary display.
<i>Type</i>	Returns or sets the alarm type. Appears in the Alarm column on the Alarm Summary display.
<i>Units</i>	Returns or sets the alarm units, which are used to interpret the <i>value</i> property.
<i>Value</i>	Returns or sets the alarm value.

Methods

None.

Events

None.

Example

This example creates an AlarmDetails object, and then uses it to define the alarm's properties before raising the alarm.

```
Set AlMDtls = Server.CreateAlarmDetails
AlMDtls.Description = "Valve has been closed."
AlMDtls.Priority = hscJournal
AlMDtls.Area = "B3"
Server.GenerateAlarm AlMDtls
```

Example scenarios

- “E-mail response for unacknowledged alarms” on page 118
- “Raising an urgent alarm when related points go into alarm” on page 127
- “Changing the ranges of related points” on page 111
- “Checking a point's value after it has gone into alarm” on page 113
- “Setting a point to a specific value at the start of each day” on page 133

Related topics

- “Creating and raising alarms” on page 22

Alert object**Description**

The Alert object represents the alert that caused the current script to run.

Properties

Property	Description
<i>Condition</i>	Returns the name of the condition which fired the event. The condition appears in the Condition column on the Alert Summary display.
<i>Source</i>	Returns the name of the source which fired the event. The source appears in the Source column on the Alert Summary display.

Remarks

- The “EventInfo object” on page 74 provides additional information about the alert.
- The “Alerts collection” on page 72 represents all alerts.

Methods

None.

Events

“OnAlert event” on page 97

“OnAlertAcknowledge event” on page 97

“OnAlertNormal event” on page 98

Example scenarios

“Generating a report in response to an alert” on page 119

“Raising an alarm in response to an alert” on page 126

Alerts collection**Description**

The collection for “Alert object” on page 71s.

Properties

Property	Description
<i>Item(source.condition)</i>	Returns the Alert object with the specified source and condition. The source and condition appear in the Source and Condition column on the Alert Summary display.
<i>server</i>	Returns the “Server object” on page 83.

Remarks

- For an introduction to collections, see “Collections” on page 17.

Methods

None.

Events

None.

Example

This example returns the alert object for which the source is 'Tank7' and the condition is 'HighTemperature.'

```
x = Alerts("Tank7.HighTemperature")
```

Area object**Description**

The Area object represents an assignable asset in the server database.

Properties

Property	Description
<i>Number</i>	Returns the sequential number assigned to the asset. (The display <i>sys137</i> shows the number assigned to each asset.) The number of the system area is <i>1</i> .
<i>Name</i>	Returns the asset name. The name will match the tag name of the assignable asset.
<i>Description</i>	Returns the descriptive title of the asset.
<i>AlarmCount(type)</i>	Returns the number of alarms for the asset, based on <i>type</i> : <ul style="list-style-type: none"> • <i>hscAlarmCountAckUrgent</i> • <i>hscAlarmCountAckHigh</i> • <i>hscAlarmCountAckLow</i> • <i>hscAlarmCountAll</i> • <i>hscAlarmCountAllAck</i> • <i>hscAlarmCountAllUrgent</i> • <i>hscAlarmCountAllHigh</i> • <i>hscAlarmCountAllLow</i> • <i>hscAlarmCountAllUnack</i> • <i>hscAlarmCountUnackUrgent</i> • <i>hscAlarmCountUnackHigh</i> • <i>hscAlarmCountUnackLow</i>
<i>Server</i>	Returns the Server object.

Remarks

- The “Areas collection” on page 73 represents assignable assets.

Methods

None.

Events

None.

Example scenarios

“Silencing a siren without acknowledging the alarm” on page 134

Areas collection

Description

The collection for Area objects.

Properties

Property	Description
<i>Count</i>	Returns the number of assignable assets that are configured on the server.

Property	Description
<i>Item(assettagname)</i>	Returns the Area object for the given <i>assettagname</i> , which can be specified as either the assignable asset tag name, full item name or the asset number in the areas collection. (The asset detail display shows both the area number and the area code of the asset.) This property only returns assignable assets.
<i>Server</i>	Returns the Server object.

Remarks

- For an introduction to collections, see “Collections” on page 17.

Methods

None.

Events

None.

Example

This example shows how you can use Areas collection to perform an iterative task. In this case, the description and number of unacknowledged alarms for each assignable asset is obtained. (This data could then be written to a file.)

```
For Each tempArea In Areas
AreaName = tempArea.Description
AlarmCount = tempArea.Count(hscAlarmCountAllUnack)
.
.
Next
```

Example scenarios


“Silencing a siren without acknowledging the alarm” on page 134

EventInfo object**Description**

The EventInfo object represents the event that caused the current script to run.

Properties

Property	Description
<i>Area</i>	<p>Returns the tag name of the assignable asset to which the event belongs.</p> <p>Only applicable to:</p> <ul style="list-style-type: none"> • OnChange event and OnOperChange event for the Param object • OnAcknowledge event, OnAlarm event, OnAlert event, OnAlertAcknowledge event, OnAlertNormal event, and OnNormal event for the Point object <p>Returns an 'empty' or 'blank' string if not applicable.</p>

Property	Description
<i>Description</i>	<p>Returns the event's description.</p> <p>Only applicable to:</p> <ul style="list-style-type: none"> OnAcknowledge event, OnAlarm event, OnAlert event, OnAlertAcknowledge event, OnAlertNormal event, and OnNormal event for the Point object OnOperChange event for the Param object <p>Returns an 'empty' or 'blank' string if not applicable.</p>
<i>Period</i>	<p>Returns the period if the event is an OnTimer event.</p> <p>Returns 0 if not applicable.</p>
<i>Priority</i>	<p>Returns the event's priority:</p> <ul style="list-style-type: none"> <i>hscJournal</i> <i>hscLow</i> <i>hscHigh</i> <i>hscUrgent</i> <p>Only applicable to an OnAcknowledge event, OnAlarm event, OnAlert event, OnAlertAcknowledge event, OnAlertNormal event, and OnNormal event for the Point object.</p> <p>Note that alerts always return <i>hscLow</i>.</p>
<i>Quality</i>	<p>Only applicable to an OnChange event for the Param object. Returns the parameter's quality:</p> <ul style="list-style-type: none"> <i>hscParamQualityGood</i>= the parameter's value is OK <i>hscParamQualityStale</i>= the parameter's value is stale <i>hscParamQualityBad</i>= the parameter's value is bad, or an error was encountered when attempting to read or write the parameter <i>hscParamQualityConfigError</i>= either the point or the parameter is invalid <i>hscParamQualityLastKnown</i>= the server system has stopped <i>hscParamQualityOutOfService</i>= the server is running as a backup (only applicable to a redundant server system) <hr/> <p> Attention</p> <ul style="list-style-type: none"> It is difficult to associate a quality with some types of points because not all points have an 'on scan' attribute. In order to find out if a standard point is on scan, check the OnScan parameter. In the case of a process point, check the EXECSTAT of the Control Module to check whether it is 'in service.'
<i>Time</i>	<p>Returns the date and time at which the event occurred.</p>
<i>TimerID</i>	<p>Returns the timer ID for a timer event.</p> <p>Only applicable to the OnTimer event.</p> <p>Returns 0 if not applicable.</p>
<i>Type</i>	<p>Returns the:</p> <ul style="list-style-type: none"> Alarm type if an OnAcknowledge event, OnAlarm event, and OnNormal event is generated for the Point object Condition if an OnAlert event, OnAlertAcknowledge event, or OnAlertNormal event is generated Station number if an OnOperChange event is generated and the Station uses Station-based security The operator account if an OnOperChange event is generated and the Station uses operator-based security <p>Returns an 'empty' or 'blank' string if not applicable.</p>

Property	Description
<i>Units</i>	<p>Returns the units of the parameter value.</p> <p>Only applicable to:</p> <ul style="list-style-type: none"> OnOperChange event and OnChange event for the Param object OnAcknowledge event, OnAlarm event, OnAlert event, OnAlertAcknowledge event, OnAlertNormal event, and OnNormal event for the Point object <p>Returns an 'empty' or 'blank' string if not applicable.</p>
<i>Value</i>	<p>Returns the point parameter's value if the event is generated for a (point) alarm, alert, or change.</p> <p>Only applicable to:</p> <ul style="list-style-type: none"> OnOperChange event and OnChange event for the Param object OnAcknowledge event, OnAlarm event, OnAlert event, OnAlertAcknowledge event, OnAlertNormal event, and OnNormal event for the Point object <p>Returns an 'empty' string if not applicable.</p>

Remarks

- The EventInfo object allows you to get more information about the event and the object to which it is attached. The information that can be obtained depends on both the event and the object.
- In the case of an alert-related event, the Alert object provides additional information about the alert.

Methods

None.

Events

None.

Example

This example uses the Area property to determine whether the event is assigned to asset 'A2.'

```
If strComp("A2",EventInfo.Area, vbTextCompare)= 0 Then
    'Perform task for area A2.
    .
    .
End If
```

This example uses the *TimerID* property to determine which timer caused the *onTimer* event.

```
If (EventInfo.TimerID = 1) Then
    'perform task associated with timer 1
    .
    .
ElseIf (EventInfo.TimerID = 2) Then
    'perform task associated with timer 2
    .
End If
```

Example scenarios

“Capturing and storing data in a text file” on page 108

“Changing a report's frequency” on page 109

“Changing the ranges of related points” on page 111

Related topics

“Simulating events” on page 44

Param object**Description**

The Param object represents a point parameter in the server database.

Properties

Property	Description
<i>EventInfo</i>	Returns the EventInfo object.
<i>Name</i>	Returns the parameter's name.
<i>ParamValue</i> ("point.param")	<p>Returns or sets the value of the specified point parameter.</p> <hr/> <p>! Attention</p> <ul style="list-style-type: none"> Some parameters, such as the PV of standard point, are read-only. <p>If a parameter's values have been enumerated, ParamValue returns/sets the enumerated value.</p>
<i>Point</i>	Returns the Point object.
<i>Quality</i>	<p>Returns the parameter's quality:</p> <ul style="list-style-type: none"> <i>hscParamQualityGood</i> = the parameter's value is OK <i>hscParamQualityStale</i> = the parameter's value is stale <i>hscParamQualityBad</i> = the parameter's value is bad, or an error was encountered when attempting to read or write the parameter <i>hscParamQualityConfigError</i> = either the point or the parameter is invalid <i>hscParamQualityLastKnown</i> = the server system has stopped <i>hscParamQualityOutOfService</i> = the server is running as a backup (only applicable to a redundant server system) <hr/> <p>! Attention</p> <p>It is difficult to associate a quality with some types of points because the concept of a point being 'on scan' is not an attribute of all point types. In order to find out if a standard point is on scan, check the OnScan parameter. In the case of a process point, check the EXECSTAT of the Control Module to check whether it is 'in service'.</p>
<i>Value</i>	<p>Returns or sets the value of this point parameter. Use the ParamValue property to read/write to another point parameter.</p> <p>If a parameter's values have been enumerated, Value returns/sets the enumerated value.</p>
<i>ValueOrdn</i>	<p>Returns or sets the parameter's state. (Value returns/sets the enumerated value.)</p> <p>Only applicable if the parameter value is enumerated.</p>

Remarks

- See the *Server and Client Configuration Guide* for a description of the parameters for standard points.
- The Params collection represents all parameters for the specified point.

Methods

“CommitValue method” on page 85

“FireScriptEvent method” on page 88

Events

“OnChange event” on page 98

“OnOperChange event” on page 100

ParamBlock object**Description**

A ParamBlock represents a *parameter block* in the server database. A parameter block allows you to pass parameters to a task, which can be either a standard server program, such as the Server Display program (LRN 21), or another application.

Properties

Property	Description
<i>Crt</i>	When used with certain standard server programs, this represents the Station number. For example, in the case of the Server Display program, it specifies the Station on which the task is performed.
<i>Param1</i> <i>Param2</i> <i>Param3</i> <i>Param4</i>	These are 16-bit (Int2) integers.
<i>Path</i>	When used with certain standard server programs, this represents the associated filename.

Remarks

- The content requirements of a parameter block are specific to the task to which the parameter block is passed. For details about the Server Display Program (also known as 'LRN 21'), see the *Server and Client Configuration Guide*. For details about writing your own applications, see the *Application Development Guide*.

Methods

None.

Events

None.

Example

This example creates a parameter block called 'PrmBlk' and defines the parameter values. After the values have been defined, the parameter block is passed to a task (LRN 111) for execution.

```
Dim PrmBlkSet
PrmBlk = Server.CreateParamBlock
'specify the parameter values
:
:
Server.RequestTask 111,PrmBlk
```

Example scenarios

“Calling up a display on a particular Station” on page 107

“Sending a report to a particular Station” on page 131

“Requesting a task when two points are enabled” on page 130

Params collection

Description

The collection for Param objects for the specified point.

Properties

Property	Description
<i>Point</i>	Returns the Point object.
<i>Item(parameter)</i>	Returns the Param object that represents the specified <i>parameter</i> .

Remarks

- For an introduction to collections, see “Collections” on page 17.

Methods

None.

Events

None.

Point object

Description

The Point object represents a point in the server database.

Properties

Property	Description
<i>EventInfo</i>	Returns the EventInfo object.
<i>Name</i>	Returns the point's name.
<i>Params</i>	Returns the Params collection for the point.
<i>ParamValue("point.param")</i> or <i>ParamValue(".param")</i>	<p>Returns or sets the value of a point parameter. Use:</p> <ul style="list-style-type: none"> <i>point.param</i> to specify the point and parameter where <i>point</i> is the point ID or the full item name of the point <i>.param</i> to specify the parameter for the current point <p>If the parameter is enumerated, ParamValue returns/sets the enumerated value.</p> <p>See the <i>Server and Client Configuration Guide</i> for a description of the parameters for standard points.</p> <p>Some parameters, such as the PV of status point, are read-only.</p>
<i>Server</i>	Returns the Server object.

Remarks

- The Points collection object represents all points.

Methods

“CommitValue method” on page 85

“CreateTimer method” on page 87

“FireScriptEvent method” on page 88

“KillTimer method” on page 91

“ModifyTimer method” on page 92

“TimerExists method” on page 95

Events

“OnAcknowledge event” on page 96

“OnAlarm event” on page 96

“OnAlert event” on page 97

“OnAlertAcknowledge event” on page 97

“OnAlertNormal event” on page 98

“OnNormal event” on page 99

“OnTimer event” on page 101

Points collection**Description**

The collection for Point objects.

Properties

Property	Description
<i>Item(pointID)</i>	Returns the Point object, where <i>pointID</i> is the point's ID or full item name.
<i>Server</i>	Returns the Server object.

Remarks

- For an introduction to collections, see “Collections” on page 17.
- The Points collection is not enumerated because of the potential performance problems this could cause on a system with many thousands of points.

Methods

None.

Events

None.

Report object

Description

The Report object represents a report.

Properties

Property	Description
<i>EventInfo</i>	Returns the “EventInfo object” on page 74.
<i>Name</i>	Returns the name of the report.
<i>Number</i>	Returns the report's number. (The Reports display shows the number assigned to each report.)
<i>Period</i>	Only applicable to periodic reports. Returns the report's period.
<i>Periodic</i>	Returns a value indicating whether the report is periodic: <ul style="list-style-type: none"> <i>False</i> = Non-periodic <i>True</i> = Periodic
<i>Server</i>	Returns the “Server object” on page 83.
<i>Requestable</i>	Returns a value indicating whether the report is requestable: <ul style="list-style-type: none"> <i>False</i> = Not requestable <i>True</i> = Requestable
<i>Type</i>	Returns a value indicating the report's type: <ul style="list-style-type: none"> <i>hscReportTypeNone</i> <i>hscReportTypeFreeFormat</i> <i>hscReportTypeAlarmEvent</i> <i>hscReportTypeDuration</i> <i>hscReportTypeSequenceOfEvents</i> <i>hscReportTypeCrossReference</i> <i>hscReportTypePointAttribute</i> <i>hscReportTypeDowntimeAnalysis</i> <i>hscReportTypeHistoryArchive</i> <i>hscReportTypeMicrosoftExcel</i> <p>Note that scripts must fully qualify the Type property, for example use: <i>Server.Reports(5).Type.</i></p>

Remarks

- The “Reports collection” on page 82 represents all reports.

Methods

“CreateTimer method” on page 87

“FireScriptEvent method” on page 88

“KillTimer method” on page 91

“ModifyTimer method” on page 92

“Request method” on page 93

“TimerExists method” on page 95

Events

“OnCompletion event” on page 99

“OnTimer event” on page 101

Reports collection**Description**

The collection for “Report object” on page 81.

Properties

Property	Description
<i>Item(n)</i>	Returns the “Report object” on page 81, where <i>n</i> is the report number. (The Reports display shows the number assigned to each report.)
<i>Count</i>	Returns the number of reports that are configured on the server.
<i>Server</i>	Returns the “Server object” on page 83.

Remarks

- For an introduction to collections, see “Collections” on page 17.

Methods

None.

Events

None.

Example scenarios

“Changing a report's frequency” on page 109

“Generating and e-mailing reports” on page 120

Script object**Description**

The Script object represents the currently executing script.

Properties

Property	Description
<i>Timeout</i>	Returns or sets the timeout period (seconds) for the script engine. If the timeout is set, the timeout returns to the standard value when the current script completes.

Remarks

- The Script object is only available to the currently executing script. It is not available to external applications, such as a VB application.

Methods

None.

Events

None.

Example

This example sets the timeout for the script to 50 seconds so that the script has time to run.

```
Sub Server_OnTimer()
    Script.Timeout = 50
    'Do long task
    .
    .
End Sub
```

Related topics

“Script engine time-outs” on page 58

“Configuring and managing automatic script engines” on page 62

“Configuring and managing manual script engines” on page 64

“Avoiding operator interaction” on page 26

Server object**Description**

The Server object is the top-level object in the object hierarchy. It represents the Application object, and contains all other objects in the object model.

Properties

Property	Description
<i>Areas</i>	Returns the Areas collection.
<i>EventInfo</i>	Returns the EventInfo object.
<i>ParamValue("point.param")</i>	<p>Returns or sets the value of a point parameter where <i>point</i> is either the point ID or the full item name of the point.</p> <hr/> <p>! Attention Some parameters, such as the PV of standard point, are read-only.</p> <p>If a parameter is enumerated, ParamValue returns/sets the enumerated value.</p>
<i>Points</i>	Returns the Points collection.
<i>Reports</i>	Returns the Reports collection.
<i>StartOfDay</i>	Returns the start of day, as specified on your system. The start of day is returned in the format specified on the server's Regional Settings.

Property	Description
<i>Status</i>	<p>Returns the server's status:</p> <ul style="list-style-type: none"> • <i>hscServerStatusUnknown</i> (or 0) • <i>hscServerStatusRunning</i> (or 1) • <i>hscServerStatusPrimary</i> (or 2) • <i>hscServerStatusBackup</i> (or 4) • <i>hscServerStatusSynchronized</i> (or 8) <p>The last three statuses are only applicable to a redundant server system.</p> <p>The status is bit pattern. For example if you have a redundant server with the servers synchronized and the call to <code>Server.Object</code> was made on the primary, you would expect to see <i>11</i> (= 1 + 2 + 8)</p>
<i>Version</i>	Returns the server's release number, for example: "100.0. "

Remarks

- The following remarks are applicable if the Server object is being accessed from another application:
 - You must only create one Server object and cache its existence, although you can make copies of it.
 - The Server object can only be created on a server if the database is loaded.
 - Due to its global nature the Server object can be referenced from any script.

Methods

"CreateAlarmDetails method" on page 86

"CreateParamBlock method" on page 87

"CreateTimer method" on page 87

"CommitValue method" on page 85

"FireScriptEvent method" on page 88

"GenerateAlarm method" on page 89

"KillTimer method" on page 91

"LogMessage method" on page 92

"ModifyTimer method" on page 92

"RequestTask method" on page 93

"SendFileToBackup method" on page 94

"TimerExists method" on page 95

Events

"OnDayStart event" on page 99

"OnShutdown event" on page 100

"OnStartup event" on page 101

"OnTimer event" on page 101

Example scenarios

"Changing the ranges of related points" on page 111

Methods

The methods are listed in alphabetical order.

Related topics

- “CommitValue method” on page 85
- “CreateAlarmDetails method” on page 86
- “CreateParamBlock method” on page 87
- “CreateTimer method” on page 87
- “FireScriptEvent method” on page 88
- “GenerateAlarm method” on page 89
- “GenerateMessage method” on page 90
- “KillTimer method” on page 91
- “LogMessage method” on page 92
- “ModifyTimer method” on page 92
- “Request method” on page 93
- “RequestTask method” on page 93
- “SendFileToBackup method” on page 94
- “TimerExists method” on page 95

CommitValue method

Applicable to

- “Param object” on page 77
- “Point object” on page 79
- “Server object” on page 83

Description

Writes the changes made to the specified point parameter(s) to disk. In a redundant server system, the value is also immediately written to the backup server, rather than at the next checkpoint period.

Syntax

For a single point parameter:

```
CommitValue "Point.Parameter"
```

For an array of point parameters:

```
CommitValue Array("Pointx.Parameterx",  
"Pointy.Parametery")
```

Part	Description
<i>Point.Parameter</i>	The point and parameter where <i>point</i> is either the point ID or full item name.

Return values

None.

Remarks

- CommitValue should be used sparingly because overuse may degrade server performance and, in the case of a redundant server system, will increase network traffic. (It is more efficient to allow values to be simultaneously checkpointed to disk.)

Example scenarios

“Working with server redundancy” on page 137

Related topics

“Server redundancy issues” on page 28

CreateAlarmDetails method**Applicable to**

“Server object” on page 83

Description

Creates an AlarmDetails object.

Syntax

```
CreateAlarmDetails
```

Return values

An AlarmDetail object.

Example

This example creates an AlarmDetails object, and then uses it to define the alarm's properties before raising (generating) the alarm.

```
Set AlmDtls = Server.CreateAlarmDetails
AlmDtls.Description = "Valve has been closed."
AlmDtls.Priority = hscJournal
AlmDtls.Area = "B3"
Server.GenerateAlarm AlmDtls
```

Example scenarios

“Raising an urgent alarm when related points go into alarm” on page 127

“Changing the ranges of related points” on page 111

“Checking a point's value after it has gone into alarm” on page 113

“Setting a point to a specific value at the start of each day” on page 133

Related topics

“Creating and raising alarms” on page 22

CreateParamBlock method

Applicable to

“Server object” on page 83

Description

Creates a ParamBlock object.

Syntax

```
CreateParamBlock
```

Return values

A ParamBlock object.

Example

This example uses a ParamBlock called 'PrmBlk' to pass a number of parameters to a task. After PrmBlk is created, its parameters are defined, before being sent to the task.

```
Dim PrmBlk
Set PrmBlk = Server.CreateParamBlock
'specifies the parameter values
PrmBlk.Param1 = 5
.
.
Server.RequestTask 111,PrmBlk
```

Example scenarios

“Requesting a task when two points are enabled” on page 130

CreateTimer method

Applicable to

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Creates a timer for an object, which then fires timer events for that object at regular intervals.

Syntax

```
CreateTimer timerID, period, starttime
```

Part	Description
<i>timerID</i>	A number that identifies the timer. The number only needs to be unique for the object—that is, two timers can have the same number, providing they are for different objects.

Part	Description
<i>period</i>	<p>The interval, in seconds, at which the timer fires timer events.</p> <p>The minimum period is <i>1</i> second.</p> <p>To minimize the load on the system, you should make the period as long as possible—especially if you create many timers.</p>
<i>starttime</i>	<p>The date/time at which the first timer event fires.</p> <p>If the value is <i>0</i>, the first event fires at the end of the first period.</p> <p>If the value is in the past, then a suitable multiple of <i>period</i> is added to it to bring it to the future.</p>

Return values

None.

Remarks

- If the object has several timers, you use the EventInfo object to determine which timer fired the event.
- You can change a timer's settings using the ModifyTimer method.
- A timer keeps firing events until it is killed by either the KillTimer method, or if the server shuts down.

Example

This example creates a timer for a point called 'Tank5.' The timer's ID is 3 and its period is 300 seconds.

```
Server.Points("Tank5").CreateTimer 3,300,0
```

This example creates a timer for a point called 'Pump4.' The timer's ID is 1, its period is 120 seconds, and the first timer event fires at 3 pm.

```
Server.Points("Pump4").CreateTimer 1,120,#3:00:00 PM#
```

Example scenarios

“Changing a report's frequency” on page 109

“Silencing a siren without acknowledging the alarm” on page 134

“Checking a point's value after it has gone into alarm” on page 113

“E-mail response for unacknowledged alarms” on page 118

Related topics

“Timers” on page 15

FireScriptEvent method**Applicable to**

“Param object” on page 77

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Simulates an event for an object. The method is typically used for testing purposes—for example, to test the server's OnStartup script without stopping and restarting the server.

This method sends the current values of the EventInfo object to the event handler. In practice, you define the simulated event's details (using the EventInfo object) before calling this method.

Syntax

FireScriptEvent *eventcode*

Part	Description
<i>eventcode</i>	<p>The code for the event that is simulated.</p> <p>For the Param object:</p> <ul style="list-style-type: none"> • <i>hscParamEventOnChange</i> • <i>hscParamEventOnOperChange</i> <p>For the Point object:</p> <ul style="list-style-type: none"> • <i>hscPointEventOnAcknowledge</i> • <i>hscPointEventOnAlarm</i> • <i>hscPointEventOnNormal</i> • <i>hscPointEventOnTimer</i> <p>For the Report object:</p> <ul style="list-style-type: none"> • <i>hscReportEventOnCompletion</i> • <i>hscReportEventOnTimer</i> <p>For the Server object:</p> <ul style="list-style-type: none"> • <i>hscServerEventOnDayStart</i> • <i>hscServerEventOnShutdown</i> • <i>hscServerEventOnStartup</i> • <i>hscServerEventOnTimer</i>

Return values

None.

Remarks

- For guidelines about using this method, see “Simulating events” on page 44.
- Automation server clients can call this method.
- This method only simulates events within scripting—the simulated events are not sent to automation server clients.

Related topics

“Simulating events” on page 44

GenerateAlarm method**Applicable to**

“Server object” on page 83

Description

Raises (generates) an alarm.

For an introduction to creating and raising alarms, see “Creating and raising alarms” on page 22.

Syntax

GenerateAlarm *alarmdetailobject*

Part	Description
<i>alarmdetailobject</i>	The alarm being raised (generated).

Return values

None.

Example

This example creates an AlarmDetails object, and then uses it to define the alarm's properties before raising (generating) the alarm.

```
Dim AlMDtls
Set AlMDtls = Server.CreateAlarmDetails
AlMDtls.Name = "Tank7"
AlMDtls.Description = "Tank overflow"
AlMDtls.Priority = hscUrgent
AlMDtls.Area = "A3"
Server.GenerateAlarm AlMDtls
```

Example scenarios

“Raising an urgent alarm when related points go into alarm” on page 127

“Changing the ranges of related points” on page 111

“Checking a point's value after it has gone into alarm” on page 113

“Setting a point to a specific value at the start of each day” on page 133

GenerateMessage method**Applicable to**

“Server object” on page 83

Description

Generates a message, which appears the Message Summary display.

Syntax

GenerateMessage *pointId, message, confirm*

Part	Description
<i>pointID</i>	The name of the point with which the message is associated. Use an empty string (") if the message does not apply to a point.
<i>message</i>	The text of the message. 80 characters maximum.

Part	Description
<i>confirm</i>	Set to <i>TRUE</i> for a confirmable message—that is a message the operator must confirm as well as acknowledge. Set to <i>FALSE</i> for an informational message.

Return values

None.

Example

This example generates a message if the PV of a point called 'Motor3' changes to 'Stopped.'

```
If StrComp(ParamValue("motor3.PV"), "Stopped", vbTextCompare) = 0 Then
    Server.GenerateMessage "motor3", "Motor has stopped.", FALSE
End if
```

Example scenarios

“Changing a report's frequency” on page 109

“Generating a report in response to an alert” on page 119

“Generating messages in Station” on page 121

KillTimer method**Applicable to**

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Kills the specified timer.

Syntax

`KillTimer timerID`

Part	Description
<i>timerID</i>	The ID of the timer that is killed.

Return values

None.

Example scenarios

“Changing a report's frequency” on page 109

“E-mail response for unacknowledged alarms” on page 118

“Checking a point's value after it has gone into alarm” on page 113

“Checking a value at regular intervals” on page 115

Related topics

“Timers” on page 15

LogMessage method**Applicable to**

“Server object” on page 83

Description

Logs a message to the server’s error log.

Syntax

LogMessage *message*

Part	Description
<i>message</i>	The message. Maximum 1,000 characters.

Return values

None.

ModifyTimer method**Applicable to**

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Modifies the specified timer.

Syntax

ModifyTimer *timerID, period, startdate*

Part	Description
<i>object</i>	The object whose timer is being modified.
<i>timerID</i>	The ID of the timer that is being modified.
<i>period</i>	The timer's period, in seconds. The minimum period is 1 second.
<i>startdate</i>	The date/time at which the first timer event fires. If the value is in the past, then a suitable multiple of <i>period</i> is added to it to bring it to future. If the value is 0, the first event fires at the end of the first period.

Return values

None.

Example

This example modifies timer 3 for a point called 'Tank5,' and sets its period to 480 seconds (8 minutes).

```
Points("Tank5").ModifyTimer 3,480,0
```

Related topics

“Timers” on page 15

Request method

Applicable to

“Report object” on page 81

Description

Requests a report (specified by the report number). The report is run by the system at the next time boundary.

Syntax

```
Request
```

Return values

None. (The request returns without waiting for the report to run or complete.)

Remarks

- The report's number is shown in the **Reports** display.
- For the report to run, you must define the **Destination** and **Operator ID** in the **Periodic Reporting** section of the **Report Configuration** display. (The **Operator ID** determines which assets are included in the report.)

Example

This example requests the tenth report in the Reports collection.

```
Reports(10).Request
```

Example scenarios

“Generating and e-mailing reports” on page 120

“Changing a report's frequency” on page 109

RequestTask method

Applicable to

“Server object” on page 83

Description

Requests a task, identified by its LRN.

Syntax

`RequestTask lrn, paramblock`

Part	Description
<i>lrn</i>	The LRN of the task.
<i>paramblock</i>	The parameter block that is passed to the task. (This contains arguments/values required by the task.)

Return values

None. (The request returns without waiting for the task to run or complete.)

Remarks

- The CreateParamBlock method is used to create a ParamBlock object, which is then used to define the values of the parameters before they are passed to the task.

Example

This example creates a parameter block called 'PrmBlk' and defines the parameter values. After the values have been defined, the parameter block is passed to a task (LRN 111) for execution.

```
Dim PrmBlk
Set PrmBlk = Server.CreateParamBlock
'specify the parameter values
'.
'.Server.RequestTask 111,PrmBlk
```

Example scenarios

“Requesting a task when two points are enabled” on page 130

SendFileToBackup method**Applicable to**

“Server object” on page 83

Description

Sends the specified file to the backup server in a redundant server system.

Syntax

`SendFileToBackup filename`

Part	Description
<i>filename</i>	The name of the file, including the extension, that is sent to the backup server.

Return values

None.

TimerExists method

Applicable to

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Checks for the existence of the specified timer.

Syntax

`TimerExists(timerID)`

Part	Description
<i>timerID</i>	The ID of the timer that is being modified.

Return values

True if a timer with the specified ID exists for the object. Otherwise *False*.

Example

This example creates timer 3 for a point called 'Tank5' if the timer hasn't already been created.

```
If Points('Tank5').TimerExists(3) = false Then
    Points('Tank5').CreateTimer 3, 480, 0
End If
```

Related topics

“Timers” on page 15

Events

The events are listed in alphabetical order.

Related topics

- “OnAcknowledge event” on page 96
- “OnAlarm event” on page 96
- “OnAlert event” on page 97
- “OnAlertAcknowledge event” on page 97
- “OnAlertNormal event” on page 98
- “OnChange event” on page 98
- “OnCompletion event” on page 99
- “OnDayStart event” on page 99
- “OnNormal event” on page 99
- “OnOperChange event” on page 100
- “OnShutdown event” on page 100
- “OnStartup event” on page 101
- “OnTimer event” on page 101

OnAcknowledge event

Applicable to

“Point object” on page 79

Description

Fires when an operator acknowledges an alarm on a point.

Remarks

- The application through which you write a script for this event depends on the point type. For a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station

OnAlarm event

Applicable to

“Point object” on page 79

Description

Fires when a point goes into alarm.

Remarks

- An OnAlarm event fires if a point is in alarm when it is created/modified.
- An OnAlarm event does not fire for a non-latched alarm (that is, the alarm condition is automatically removed when the point returns to its normal state). To perform scripting on such a point, use the “OnNormal event” on page 99.

- The application through which you write a script for this event depends on the point type. For a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station

Example scenarios

“Disabling and enabling alarms for related points” on page 117

“Raising an urgent alarm when related points go into alarm” on page 127

“Changing the ranges of related points” on page 111

“Checking a point's value after it has gone into alarm” on page 113

OnAlert event

Applicable to

“Alert object” on page 71

“Point object” on page 79

Description

Fires when an alert is generated.

Remarks

- An OnAlert event fires if an alert is active when it is created/modified.
- The “EventInfo object” on page 74 provides information about the event, in addition to the Alert object.
- You write a script for this event in Station, except for alerts generated by process points. You write a script for process point alerts in Control Builder.

Example scenarios

“Generating a report in response to an alert” on page 119

“Raising an alarm in response to an alert” on page 126

OnAlertAcknowledge event

Applicable to

“Alert object” on page 71

“Point object” on page 79

Description

Fires when an operator acknowledges an alert.

Remarks

- The “EventInfo object” on page 74 provides information about the event, in addition to the Alert object.
- You write a script for this event in Station, except for alerts generated by process points. You write a script for process point alerts in Control Builder.

Example scenarios

“Generating a report in response to an alert” on page 119

OnAlertNormal event

Applicable to

“Alert object” on page 71

“Point object” on page 79

Description

Fires when an alert returns to normal.

Remarks

- The “EventInfo object” on page 74 provides information about the event, in addition to the Alert object.
- You write a script for this event in Station, except for alerts generated by process points. You write a script for process point alerts in Control Builder.

Example scenarios

“Generating a report in response to an alert” on page 119

OnChange event

Applicable to

“Param object” on page 77

Description

Fires when the value or quality of a point parameter is changed.

Remarks

- If the change was made by a Station operator, an “OnOperChange event” on page 100 event also fires.
- Notification of an OnChange event is not immediate—the default period at which the system checks for such changes is 1 second.
- If many parameters have OnChange events, performance problems may occur.
- If there are two change events that occur in the same second that counteract each other, no events are fired. For example, an OnChange event is attached to the TotalActive Alarms parameter for an alarm group. In the same second, an alarm is raised and an alarm returns to normal. The totalActiveAlarms count does not change and therefore no event is fired.
- The application through which you write a script for this event depends on the point type to which the parameter belongs. For a parameter of a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station

Example scenarios

“Counting the number of pump starts” on page 116

“Capturing and storing data in a text file” on page 108

“Changing a set point parameter with a library function” on page 110

OnCompletion event

Applicable to

“Report object” on page 81

Description

Fires when the report has been created (generated).

Remarks

- You write a script for this event through Station.

Example scenarios

“Generating and e-mailing reports” on page 120

OnDayStart event

Applicable to

“Server object” on page 83

Description

Fires at the start of a new day (as configured on your system).

Remarks

- You write a script for this event through Station.

Example scenarios

“Performing calculations on a process variable” on page 124

“Setting a point to a specific value at the start of each day” on page 133

“Launching an external application” on page 123

OnNormal event

Applicable to

“Point object” on page 79

Description

Fires when an alarmed condition returns to normal.

Remarks

- The application through which you write a script for this event depends on the point type. For a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station

Example scenarios

“Checking a point's value after it has gone into alarm” on page 113

OnOperChange event**Applicable to**

“Param object” on page 77

Description

Fires when a Station operator initiates a change in the value or quality of a point parameter.

Remarks

- An operator-initiated change also fires an OnChange event—that is, two events fire: OnOperChange and OnChange.
- The application through which you write a script for this event depends on the point type to which the parameter belongs. For a parameter of a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station
- You should use `eventinfo.value` rather than `paramvalue`, `value`, or `valueordn` calls. The reason for this is that the database value will not be updated until the next time the database is scanned for the result.

Example scenarios

“E-mail response for unacknowledged alarms” on page 118

“Using an auxiliary parameter to store calculated data” on page 135

“Reading data from a text file” on page 128

“Changing a set point parameter with a library function” on page 110

OnShutdown event**Applicable to**

“Server object” on page 83

Description

Fires when the server shuts down, but before tasks are stopped. That is, it defines when the Server requests all tasks to stop, not when the tasks actually stop.

Remarks

- If the system is very busy at shutdown, the system may shut down before the corresponding scripts are run.
- You write a script for this event through Station.

Example scenarios

“Working with server redundancy” on page 137

OnStartup event

Applicable to

“Server object” on page 83

Description

Fires when the server starts (after all tasks have been invoked).

Remarks

- In a redundant server system, an OnStartup event fires when the backup server becomes the primary server.
- You write a script for this event through Station.

OnTimer event

Applicable to

“Point object” on page 79

“Report object” on page 81

“Server object” on page 83

Description

Fires at regular intervals, as determined by the associated timer.

Remarks

- If an object has more than one timer, you use the EventInfo object and its TimerID property to determine which timer generated the event.
- OnTimer events are not preserved on failover in a redundant server system.
- The application through which you write a script for this event depends on the type of object. For a:
 - Standard point, use Quick Builder
 - Process point, use Control Builder
 - Flexible point, use Station
 - Report, use Station
 - Server, use Station

Example

This example shows the OnTimer script for a point called “Pump5,” which has two timers. The task that is performed depends on the timer that generated the event.

```
Sub Pump5_OnTimer()
  If (EventInfo.TimerID = 1) Then
    'perform task associated with timer 1
    .
    .
  If (EventInfo.TimerID = 2) Then
    'perform task associated with timer 2
    .
  End If
End Sub
```

Example scenarios

“Checking a value at regular intervals” on page 115

“E-mail response for unacknowledged alarms” on page 118

“Checking a point's value after it has gone into alarm” on page 113

“Performing calculations on a process variable” on page 124

Related topics

“Timers” on page 15

Example server scripting scenarios

These examples contain scripts that perform a wide range of realistic tasks.

In general, the Script Editor hides the standard VBScript sub and End sub statements. Although the examples include these statements for reasons of completeness, the Script Editor hides them. (The statements are not hidden in library scripts.)

```
Sub Server_OnShutdown()  
    ParamValue("SHUTDOWNSTATUS.STATE") = "SHUTDOWN"  
End Sub
```

You can speed up script development by copying examples from this help. Remember to copy only the lines of code inside the sub and End sub statements and not the sub and End sub lines themselves.

Related topics

- “Copying example Script Editor scripts” on page 105
- “Calling up a display on a particular Station” on page 107
- “Capturing and storing data in a text file” on page 108
- “Changing a report's frequency” on page 109
- “Changing a set point parameter with a library function” on page 110
- “Changing the ranges of related points” on page 111
- “Checking a point's value after it has gone into alarm” on page 113
- “Checking a value at regular intervals” on page 115
- “Counting the number of pump starts” on page 116
- “Disabling and enabling alarms for related points” on page 117
- “E-mail response for unacknowledged alarms” on page 118
- “Generating a report in response to an alert” on page 119
- “Generating and e-mailing reports” on page 120
- “Generating messages in Station” on page 121
- “Journaling an event at regular intervals” on page 122
- “Launching an external application” on page 123
- “Performing calculations on a process variable” on page 124
- “Performing the same calculation on a set of points” on page 125
- “Raising an alarm in response to an alert” on page 126
- “Raising an urgent alarm when related points go into alarm” on page 127
- “Reading data from a text file” on page 128
- “Requesting a task when two points are enabled” on page 130
- “Sending a report to a particular Station” on page 131
- “Sending data to other displays” on page 132
- “Setting a point to a specific value at the start of each day” on page 133
- “Silencing a siren without acknowledging the alarm” on page 134
- “Using an auxiliary parameter to store calculated data” on page 135
- “Using auxiliary parameters to store and display data” on page 136

- “Working with server redundancy” on page 137
- “Testing and debugging scripts” on page 43
- “Server Scripting Object Model reference” on page 67
- “Writing scripts for alerts” on page 35
- “Writing scripts for flexible points” on page 34
- “Writing scripts for process points” on page 32
- “Writing scripts for standard points” on page 33
- “Script Editor basics” on page 39
- “Script Editor basics specific to Station” on page 40
- “Writing scripts for the server, library or periodic scripts” on page 36
- “Copying example Script Editor scripts” on page 105

Copying example Script Editor scripts

You can save a lot of time by copying example scripts from this help, pasting them into your own Script Editor scripts, and then making the appropriate modifications.

The following example describes how to copy a block of code and paste it into one of your Script Editor scripts.

To copy and paste a block of Script Editor code

- 1 In the help, drag diagonally across the code that you want to copy.
Do not select the `sub` and `end sub` statements because they are automatically created, but hidden from view, by the Script Editor.

```
Sub Server OnDayStart()  
    ParamValue("O1IFBLUE.A1") = 50.0  
    ParamValue("O1IFBLUE.A2") = 0  
End Sub
```

- 2 Copy the code by pressing CTRL+C.
- 3 Switch to the Script Editor.
- 4 Click where you want to insert the code (typically a blank line), and then press CTRL+V.
If you accidentally select a `sub` and `end sub` statement, the Script Editor will flag the error, and you must delete that line.

Related topics

- “Script Editor basics” on page 39
- “Example server scripting scenarios” on page 103
- “Calling up a display on a particular Station” on page 107
- “Capturing and storing data in a text file” on page 108
- “Changing a report's frequency” on page 109
- “Changing a set point parameter with a library function” on page 110
- “Changing the ranges of related points” on page 111
- “Checking a point's value after it has gone into alarm” on page 113
- “Checking a value at regular intervals” on page 115
- “Counting the number of pump starts” on page 116
- “Disabling and enabling alarms for related points” on page 117
- “E-mail response for unacknowledged alarms” on page 118
- “Generating a report in response to an alert” on page 119
- “Generating and e-mailing reports” on page 120
- “Generating messages in Station” on page 121
- “Journaling an event at regular intervals” on page 122
- “Launching an external application” on page 123
- “Performing calculations on a process variable” on page 124
- “Performing the same calculation on a set of points” on page 125
- “Raising an alarm in response to an alert” on page 126
- “Raising an urgent alarm when related points go into alarm” on page 127
- “Reading data from a text file” on page 128
- “Requesting a task when two points are enabled” on page 130
- “Sending a report to a particular Station” on page 131
- “Sending data to other displays” on page 132

- “Setting a point to a specific value at the start of each day” on page 133
- “Silencing a siren without acknowledging the alarm” on page 134
- “Using an auxiliary parameter to store calculated data” on page 135
- “Using auxiliary parameters to store and display data” on page 136
- “Working with server redundancy” on page 137

Calling up a display on a particular Station

Scenario

You want to call up a display on a particular Station each time a status point changes value.

Solution

The status point's OnChange event uses the Server Display program (LRN 21) to call up the display on the specified Station. (The Server Display Program performs numerous tasks, depending on the parameter values. For details about the Server Display Program (LRN21), see the *Server and Client Configuration Guide*.)

You create a parameter block so that you can pass the required parameters to LRN 21. In this scenario, you must specify the following.

ParamBlock Property (LRN Parameter)	Description
<i>crt</i>	The number of the Station on which the display is called up.
<i>Param1</i>	Set to <i>1</i> . (This instructs LRN 21 to call up the display specified in Path)
<i>Param2</i>	Set to <i>0</i> . (This instructs LRN 21 to call up a named display).
<i>Path</i>	<p>The name of the display.</p> <p>If the display is a HMIWeb display, the syntax is:</p> <p><i>page://pagename</i></p> <p>If the display is a DSP display, the syntax is:</p> <p><i>pagename</i></p>

Scripts

```
Sub CDAStatus.PV_OnChange()
  Dim ParamBlock
  Set ParamBlock = Server.CreateParamBlock
  ParamBlock.crt = 1
  ParamBlock.param1 = 1
  ParamBlock.param2 = 0
  ParamBlock.path = "page://MainTanks"
  Server.RequestTask 21, ParamBlock
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Capturing and storing data in a text file

Scenario

You want to capture the time and date when a pump starts and stops, and copy the data to a text file so that you can analyze it using Microsoft Excel.

Solution

The script is attached to an OnChange Event for the point attached to the pump. The comma-delimited data is written to a text file, *Pumpdata.txt*.

Scripts

```
Sub PUMP_PV_OnChange() 'Pump's pv
    dim file, filename, msgstream
    Set file = CreateObject("Scripting.FileSystemObject")
    filename= "c:\server\user\Pumpddata.txt"
    Const ForReading = 1, ForWriting = 2, ForAppending = 8

    Set msgstream = file.OpenTextFile(filename, ForAppending, TRUE)
    ' Open for writing, create if non existent.
    If (StrComp(EventInfo.value, "ON", vbTextCompare) = 0)
        Then Msgstream.WriteLine 'RUNNING " & ', ' &
            Date & " " & Time
    ElseIf (StrComp(EventInfo.value, "OFF", vbTextCompare) = 0) Then
        Msgstream.WriteLine "STOPPED " & ', ' &
            Date & ' ' & Time
    End If
    msgstream.close
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Changing a report's frequency

Scenario

Because of problems that occurs on the night shift, you want to run one of your hourly reports at 15-minute intervals. (You still want to run the report on an hourly basis for the rest of the day.)

Solution

The OnShiftStart event of the server is used to run a script that creates a 15-minute (900 second) timer for the desired report. The report is requested in the script attached to the OnTimer event of the server. Note that this script checks the error code returned by the script and raises a message if the report fails.

Scripts

The OnShiftStart script for the Server:

```
Sub Server_OnShiftStart()
    Dim TimeNow
    TimeNow = Time

    If ((TimeNow > #11:50:00 PM#) Or (TimeNow < #7:50:00
        AM#)) Then
        CreateTimer 1,900,0
    Else
        KillTimer 1
    End If
End Sub
```

The OnTimer script for the Server:

```
Sub Server_OnTimer()
    On Error Resume Next
    If (EventInfo.TimerID = 1) Then
        Reports(10).Request
        If (Err.Number <> 0) Then
            GenerateMessage "", "Failed to request report 10.", FALSE
        End if
    End If
End Sub
```

Remarks

- The report must be demandable, and must be configured as a valid report before it is requested by the script.
- The code assumes that the shift changes are at 08:00 hrs, 16:00 hrs, and 24:00 hrs.

Related topics

“Copying example Script Editor scripts” on page 105

Changing a set point parameter with a library function

Scenario

You have many points for which you want to change the SP when the PV changes. The calculation used to determine the SP is based on the new PV, and is the same for all points.

Solution

Create a library script that performs the calculation. It can then be called as required by the OnChange scripts for the PV of the relevant points.

In this example, the library script is called 'EWMA,' and is called by the OnChange script for the PV of a point called 'Tank.'

Scripts

The Library script (note that the Sub and End Sub statements are required):

```
Sub EWMA(Server, Tank, Pump)
    Server.ParamValue(Pump) = Sqr(Server.ParamValue(Tank)/5) + 0.05
End Sub
```

The OnChange script for the PV of the point, Tank:

```
Sub Tank_PV_OnChange()
    EWMA Server, "Tank.PV", "Pump.SP"
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Changing the ranges of related points

Scenario

You want to change the ranges on some points based on the product that is currently being refined—for example to cater for light crude oil, heavy crude oil and so on.

Solution

An OnAlarm event attached to a status point 'HeavyValve1' has a script that changes the Engineering Units (and potentially Set Point limits) of a group of points. These changes are also recorded in the event log.

Scripts

The OnAlarm script for the point, HeavyValve1:

```
Sub HeavyValve1_OnAlarm()
    Dim AlMDtls

    If (StrComp(EventInfo.Type, "CHANGE", vbTextCompare) = 0) Then
        If (StrComp(ParamValue(".PV"), "OPEN", vbTextCompare) = 0) Then
            Set AlMDtls = Server.CreateAlarmDetails

            ParamValue("Furnace1.EUHI") = 400.0
            ParamValue("Furnace1.EULO") = 80.0

            AlMDtls.Description = "Started processing Heavy Crude."
            AlMDtls.Priority = hscJournal
            AlMDtls.Area = ""
            Server.GenerateAlarm AlMDtls

            AlMDtls.Description = "Furnace1.EUHI adjusted."
            AlMDtls.Value = 400.0
            Server.GenerateAlarm AlMDtls

            AlMDtls.Description = "Furnace1.EULO adjusted."
            AlMDtls.Value = 80.0
            Server.GenerateAlarm AlMDtls
        End If
    End If
End Sub
```

The OnAlarm script for the point, LightValve1:

```
Sub LightValve1_OnAlarm()
    Dim AlMDtls

    If (StrComp(EventInfo.Type, "CHANGE", vbTextCompare) = 0) then
        If (StrComp(ParamValue(".PV"), "OPEN", vbTextCompare) = 0) then
            Set AlMDtls = Server.CreateAlarmDetails
            ParamValue("Furnace1.EUHI") = 600.0
            ParamValue("Furnace1.EULO") = 200.0

            AlMDtls.Description = "Started processing Light Crude."

            AlMDtls.Priority = hscJournal
            AlMDtls.Area = ""
            Server.GenerateAlarm AlMDtls

            AlMDtls.Description = "Furnace1.EUHI adjusted."
            AlMDtls.Value = 600.0
            Server.GenerateAlarm AlMDtls

            AlMDtls.Description = "Furnace1.EULO adjusted."
            AlMDtls.Value = 200.0
            Server.GenerateAlarm AlMDtls
        End if
    End If
End Sub
```

Remarks

- To swap to the other type of oil, the PV of the associated point needs to change to the 'Open' state.

Related topics

“Copying example Script Editor scripts” on page 105

Checking a point's value after it has gone into alarm

Scenario

You want to check a point's value a predetermined time after it has gone below a specified value to see whether it is still below this value or not. If it is still below that value, then you want to raise an alarm.

Solution

The point's PVLO is set to the required value. If a PVLO alarm is raised, the OnAlarm event starts a timer.

If the alarm returns to normal, the timer is killed. However, if the timer generates an event, the OnTimer event raises an alarm.

Note that error checking is performed when the timer is created and killed, and an alarm raised if the operation fails.

Scripts

The OnAlarm script for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnAlarm()
    Dim code
    Dim AlmDtls

    If (StrComp(EventInfo.Type, "PVLO", vbTextCompare) = 0)
        Then
            If (StrComp(ParamValue(".TimerStart"), "YES", vbTextCompare) <>
                0) Then
                On Error Resume Next
                CreateTimer 1,120,0
                If (Err.Number = 0) Then
                    ParamValue(".TimerStart") = "YES"
                Else
                    Set AlmDtls = Server.CreateAlarmDetails
                    AlmDtls.Description = "Failed to create a timer to monitor 01IFBLUE.PV"
                    AlmDtls.Priority = hscJournal
                    AlmDtls.Area = ""
                    Server.GenerateAlarm AlmDtls
                End If
            End If
        End If
    End Sub
```

The OnNormal script for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnNormal()
    Dim code
    Dim AlmDtls

    If (StrComp(ParamValue(".TimerStart"), "YES", vbTextCompare) = 0)
        Then
            On Error Resume Next
            KillTimer 1
            If (Err.Number = 0) Then
                ParamValue(".TimerStart") = "NO"
            Else
                Set AlmDtls = Server.CreateAlarmDetails()
                AlmDtls.Description = "Failed to kill timer 1 on point 01IFBLUE
                    after it returned to normal."
                AlmDtls.Priority = hscUrgent
                AlmDtls.Area = ""
                Server.GenerateAlarm AlmDtls
            End If
        End If
    End Sub
```

The OnTimer script for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnTimer()
    Dim code
    Dim AlmDtls
```

```

    If (EventInfo.TimerID = 1) Then
        Set AlMDtls = Server.CreateAlarmDetails
        AlMDtls.Description = "01IFBLUE has been under PVLO for
            2 minutes."
        AlMDtls.Priority = hscUrgent
        AlMDtls.Area = ""
        Server.GenerateAlarm AlMDtls

        On Error Resume Next
        KillTimer 1
        If (Err.Number = 0) Then
            ParamValue(".TimerStart") = "NO"
        Else
            AlMDtls.Description = "Failed to kill timer 1 on point 01IFBLUE."
            Server.GenerateAlarm AlMDtls
        End If
    End If
End Sub

```

Remarks

- The user-defined TimerStart parameter must be created for point 01IFBLUE as a string, and its initial value must be set to *NO*.

Related topics

“Copying example Script Editor scripts” on page 105

Checking a value at regular intervals

Scenario

When an operator changes a set point, you want wait for the process to ramp up to a specified value (which may be dependent on other process variables) before executing a particular action.

Solution

Because VBScript does not have Wait or Sleep commands, you create a timer when the operator changes the SP. The OnTimer script compares the SP and PV values each time the timer fires. (An advantage of this solution is that other scripts can run between timer events.)

Scripts

This script creates the timer for point 'XYZPOINT' when an operator changes its SP:

```
Sub XYZPOINT_SP_OnOperChange()
    Server.Points("XYZPOINT").CreateTimer 1, 10, 0
End Sub
```

The OnTimer script:

```
Sub XYZPOINT_OnTimer()
    dim desiredVal
    dim margin
    dim currentVal

    desiredVal = ParamValue(".SP")
    currentVal = ParamValue(".PV")
    margin = desiredVal*0.05

    If ((currentVal < desiredVal + margin) And (currentVal
        > desiredVal - margin)) Then
        'execute some action
        'kill the timer
        KillTimer 1
    End If
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Counting the number of pump starts

Scenario

You want to count the number of pump starts. A status point, Pump1, indicates the pump's state: Off (0) and On (1).

Solution

An accumulator point, PumpStarts, is incremented each time the pump starts.

In Quick Builder, you create a user-defined parameter for the point called 'Count.'

There are two methods of detecting a pump start:

- Attach a script on the pump's OnChange event.
- Attach a script to the pump's OnAlarm event that checks if it is a 'change' event.

Both methods use a counter, which must be reset at some time. The example in this scenario resets the counter when the server starts.

Scripts

The OnStartup script for the Server (This creates the user-defined parameter 'count' for the point 'PumpStarts' and sets it to 0 when the server starts.)

```
Sub Server_OnStartup()
    ParamValue("PumpStarts.Count") = 0
End Sub
```

Method1. The OnChange script for the PV of the point, Pump1

```
Sub Pump1_PV_OnChange()
    If StrComp(EventInfo.value, "ON", vbTextCompare) = 0
    Then
        ParamValue("PumpStarts.Count") = ParamValue("PumpStarts.Count") + 1
    End If
End Sub
```

Method2. The OnAlarm script for the point, Pump1

```
Sub Pump1_OnAlarm()
    If (StrComp(EventInfo.type, "CHANGE", vbTextCompare) = 0) Then
        If StrComp(EventInfo.value, "ON", vbTextCompare) = 0 Then
            ParamValue("PumpStarts.Count") = ParamValue("PumpStarts.Count") + 1
        End If
    End If
End Sub
```

Remarks

- Although the first method uses a slightly simpler script, it is actually less efficient because detecting point value changes involves polling the point every second to detect a change.
- In the case of method 2, configure the point, Pump1, so that alarming is enabled but do not define any of the point's states as being an alarm state. (When a status point is configured in this way, each change in state generates a Journal alarm.)

Related topics

“Copying example Script Editor scripts” on page 105

Disabling and enabling alarms for related points

Scenario

You want to disable all alarms for a set of points when a motor is stopped. You also want to enable the alarms for the points when the motor is restarted.

Solution

The script is attached to the point, 'MOTOR.' It disables alarming for the related points when the motor stops, and enables alarming for the points when the motor starts.



Attention

Be aware that when you disable alarms on a point, you will also disable any alerts on the point.

Scripts

The OnAlarm script for the point:

```
Sub MOTOR_OnAlarm()
  If (StrComp(EventInfo.value, "STOP", vbTextCompare) = 0)
    Then
      ParamValue("Point1.AlarmDisabled") = TRUE
      ParamValue("Point3.AlarmDisabled") = TRUE
      ParamValue("Point7.AlarmDisabled") = TRUE
  ElseIf (StrComp(EventInfo.value, "START", vbTextCompare) = 0) Then
      ParamValue("Point1.AlarmDisabled") = FALSE
      ParamValue("Point3.AlarmDisabled") = FALSE
      ParamValue("Point7.AlarmDisabled") = FALSE
  End If
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

E-mail response for unacknowledged alarms

Scenario

You want to e-mail or page someone if there are any unacknowledged alarms for a particular asset (Mixer) that remain unacknowledged for 15 minutes.

Solution

It is not possible to access old alarms. Consequently, the solution is as follows:

- Using the Point Script editor in Station, specify the point name or full item name of the particular asset. Create an OnChange script for the parameter corresponding to the Alarm count of interest. For example, the Unacknowledged Urgent Alarms count is stored in the 'TotalUrgentUnackAlarms' parameter.
- Have the OnChange Script start a timer. When the value of count changes from zero, set the period of the timer to the length of time it is acceptable to have an unacknowledged alarm (900 seconds in this case).
- If the count of Unacknowledged Alarms changes to zero, disable the timer.
- If the OnTimer event fires, send the e-mail or page someone.

Scripts

The OnChange script for the asset:

```
Sub Mixer.TotalUrgentUnackAlarms_OnChange()
  If (value > 0) Then
    On Error Resume Next 'Ignore the error that occurs if the timer already exists'
    Server.Points('Mixer').CreateTimer
    1, 900, 0
  Else
    Server.Points('Mixer').KillTimer
    1
  End If
End Sub
```

The OnTimer script for the asset: (Note that *NetworkNameofEmailServer* is the name of your e-mail server.)

```
Sub Mixer_OnTimer
  set msg = createobject("cdo.message")
  set iconf = createobject("cdo.configuration")
  Set flds = iconf.Fields
  With flds
    .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
    .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = NetworkNameofEmailServer
    .Update
  End With

  With msg
    Set .Configuration = iconf
    .To = "john.citizen@fudd.com.au"
    .From = "john.citizen@fudd.com"
    .Subject = "Check unacknowledged alarms"
    .TextBody = ""
    .fields.update
    .Send
  End With
End Sub
```

Remarks

- For the e-mail to be sent, the 'SMTP Server' feature of Windows Server needs to be installed. This feature requires the 'Web Server (IIS)' role to be installed. In addition, the Windows Server needs to be set up with access to a DNS server.

Related topics

“Copying example Script Editor scripts” on page 105

Generating a report in response to an alert

Scenario

You have a report that collates information about a group of related points in a section of the plant. You want to generate the report if an alert is raised with a particular source (RED02) and condition (HiTemp) in that section of plant.

Solution

You use the OnAlert event for the appropriate alert object to request the report.

Scripts

The OnAlert script for the alert source RED02:

```
Sub RED02.HiTemp_OnAlert()  
    Server.Reports(3).Request  
    If (Err.Number <> 0) Then  
        Server.GenerateMessage "", "Failed to request report 3.", FALSE  
    End if  
End Sub
```

Remarks

- The report must be demandable, and must be configured as a valid report before it is requested by the script.

Related topics

“Copying example Script Editor scripts” on page 105

Generating and e-mailing reports

Scenario

You have a report that collates important information about the plant. You want to generate this report on particular alarms, and e-mail it to yourself. (The report's number is 1, and it is configured to export the output into Microsoft Word format, therefore its filename is *rpt001.doc*.)

Solution

The OnAlarm event for the appropriate point requests the report and the report's OnCompletion script e-mails it.

Scripts

The point's OnAlarm event requests the report.

```
Sub CanteenDoor_OnAlarm()
    Server.Reports(1).Request
End Sub
```

The report's OnCompletion event sends the report. (Note that *NetworkNameofEmailServer* is the name of your e-mail server.)

```
set iMsg = createobject("cdo.message")
set iConf = createobject("cdo.configuration")
Set Flds = iConf.Fields
With Flds
    .Item("http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
    .Item("http://schemas.microsoft.com/cdo/configuration/smtpserver") = "NetworkNameofEmailServer"
    .Update
End With

With iMsg
Set .Configuration = iConf
.To = "recipient1@honeywell.com; recipient2@honeywell.com"
.From = "sender@honeywell.com"
.Subject = "VSDRe"
.AddAttachment "C:\ProgramData\Experion PKS\Server\Data\Report\rpt001.doc"
.Fields.update
.TextBody = "Any text you want in the e-mail body"
.Send
End With
```

Remarks

- This script assumes that Experion PKS is installed to the default location. Specifically, the `.AddAttachment` line in the script (`.AddAttachment "C:\..."` etc.) looks for the file in the default location. If your copy of Experion PKS is installed to a different location, remember to update this line in the script.
- For the e-mail to be sent, the 'SMTP Server' feature of Windows Server needs to be installed. This feature requires the 'Web Server (IIS)' role to be installed. In addition, the Windows Server needs to be set up with access to a DNS server.

Related topics

“Copying example Script Editor scripts” on page 105

Generating messages in Station

Scenario

You want to select and generate a message from a list of messages, based on the value of an analog point, MSG001.A2. (For example, if MSG001.A2 = 1, the message might be 'Status normal'. If MSG001.A2 = 2, the message might be 'System temperature high'.)

The messages are stored in a text file, *messages.txt*.

Solution

The script for the analog point's OnChange event generates the messages.

Scripts

The script for the OnChange event.

```
sub MSG001_A2_OnChange
    dim file, filename, msgstream,

    set file = CreateObject("Scripting.FileSystemObject")
    filename="c:\server\user\messages.txt"
    set msgstream = file.OpenTextFile(filename, ForReading, FALSE)
    ' Open for reading, do not create if non existent.
    For j = 1 to ParamValue(".A2")
        msgstream.SkipLine
    Next

    msg = msgstream.ReadLine
    file.close

    Server.GenerateMessage "", Text(msg), FALSE
end sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Journaling an event at regular intervals

Scenario

You want to journal an event every hour that records the total volume of product in five tanks.

Solution

You create a periodic script called 'ProductVolume' and set its period to 60 minutes. It adds up the levels of the five tanks and then raises a Journal priority alarm.

Scripts

```
Sub ProductVolume_OnTimer()  
    Dim AlmDtls  
  
    Set AlmDtls = Server.CreateAlarmDetails  
  
    totallevel = Server.ParamValue("Tank1.PV")  
    totallevel = totallevel + Server.ParamValue("Tank2.PV")  
    totallevel = totallevel + Server.ParamValue("Tank3.PV")  
    totallevel = totallevel + Server.ParamValue("Tank4.PV")  
    totallevel = totallevel + Server.ParamValue("Tank5.PV")  
  
    AlmDtls.Description = "Level of tank farm"  
    AlmDtls.Value = totallevel  
    AlmDtls.Priority = hscJournal  
    Server.GenerateAlarm AlmDtls  
End Sub
```

Remarks

- In addition to the period, you must define the periodic script's other properties.

Related topics

“Periodic scripts” on page 10

“Timers” on page 15

“Library scripts” on page 11

“Copying example Script Editor scripts” on page 105

Launching an external application

Scenario

You want to run a utility program once a day.

Solution

You write a script for the server's OnDayStart event that runs the utility.

Scripts

The OnDayStart script for the server:

```
Sub OnDayStart
    Dim objShell

    Set objShell = CreateObject("WScript.Shell")
    objShell.Run "C:\Utilities\MyUtility.exe"
End Sub
```

If you want to run a batch file instead of a utility the script would be:

```
Sub OnDayStart
    Dim objShell

    Set objShell = CreateObject("WScript.Shell")
    objShell.Run "cmd /C C:\temp\mybatchfile.bat"
End Sub
```

Remarks

- For details about launching and using an external application, see “Launching and interacting with external applications” on page 27.
- For details about the CreateObject function, see the help for VBScript.
- Make sure that the application does not pause and wait for input from a user.

Related topics

“Avoiding operator interaction” on page 26

“Copying example Script Editor scripts” on page 105

Performing calculations on a process variable

Scenario

You have a process variable for which you want to:

- Calculate and present the running average value of that variable as a point parameter
- Initialize the running average at the start of the day

Solution

The OnTimer event for the point (01IFBLUE) that accesses the desired process variable is used to run a script to calculate the running average value and store it into another parameter (A1). The total number of samples are stored in another parameter (A2). The average value and the number of samples are initialized by a script attached to the OnDayStart event of the server.

Scripts

The OnDayStart script for the Server:

```
Sub Server_OnDayStart()  
    ParamValue("01IFBLUE.A1") = 50.0  
    ParamValue("01IFBLUE.A2") = 0  
End Sub
```

Remarks

- See “Using auxiliary parameters to store and display data” on page 136 for the script attached to the OnTimer event of the relevant point.

Related topics

“Copying example Script Editor scripts” on page 105

Performing the same calculation on a set of points

Scenario

You want to perform a particular calculation on a set of points.

Solution

Create an array of point names, and perform the calculation on each point in the array.

Point lists can be simulated by creating an array of point names and then looping on this list performing the calculation.

Scripts

The OnChange script for the PV of the point, 01IFBLUE:

```
Sub 01IFBLUE_PV_OnChange()
  Dim TotalVolume
  Dim Volume
  Dim TankRadius
  Dim tanklevels(3)

  TotalVolume = 0
  TankRadius = 12
  tanklevels(1)="Tank1Level.PV"
  tanklevels(2)="Tank2Level.PV"
  tanklevels(3)="Tank3Level.PV"

  For each pnt in tanklevels
    Volume = 3.1416 * TankRadius * TankRadius * ParamValue(pnt)
    TotalVolume = TotalVolume + Volume
  Next

  ParamValue("Petrol.Volume") = TotalVolume
End Sub
```

Remarks

- You should consider writing library scripts for such calculations. See “Changing a set point parameter with a library function” on page 110.

Related topics

“Copying example Script Editor scripts” on page 105

Raising an alarm in response to an alert

Scenario

If a particular alert is raised, you want to check the value of a specific point and raise an alarm if the point's value is too high.

Solution

You write an OnAlert script for alert source 'Pumps1' that checks the value of point 'Valve1,' and if the value is too high, generates an urgent alarm.

Scripts

The OnAlert script for the alert source Pumps1:

```
Sub Pumps1_OnAlert()  
  Dim AlMDtls  
  If (ParamValue("Valve1.PV") > 90.00 ) Then  
    Set AlMDtls = Server.CreateAlarmDetails  
    AlMDtls.Description = "An Alert was generated on PUMPS1 and Valve1.PV is too high."  
    AlMDtls.Priority = hscUrgent  
    AlMDtls.Area = "PumpRoom"  
    Server.GenerateAlarm AlMDtls  
  End If  
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Raising an urgent alarm when related points go into alarm

Scenario

When a point goes into alarm, you want to check whether there is also an unacknowledged alarm for a related point. If the other point is also in alarm, you want to raise an urgent alarm.

(In practice, when several related points go into alarm, the reason is quite different from when just one of those points goes into alarm.)

Solution

The OnAlarm event of the point 'TANK1' runs a script that checks the 'UnackAlarmExists' status of the relevant points, and then takes the required action.

Scripts

The OnAlarm script for the point, TANK1:

```
Sub TANK1_OnAlarm()  
  Dim AlmDtls  
  
  If (ParamValue("PUMP1.UNACKALARMEXISTS")) Then  
    Set AlmDtls = Server.CreateAlarmDetails  
    AlmDtls.Description = "TANK1 and PUMP1 both have problems."  
    AlmDtls.Priority = hscUrgent  
    AlmDtls.Area = ""  
    Server.GenerateAlarm AlmDtls  
  End If  
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Reading data from a text file

Scenario

You need to calculate the specific volume of steam, given the temperature and pressure. The required 'steam table' is available as a text file on the server.

Solution

Because of the relative complexity and file handling requirements, this scenario is best solved with a custom function.

The OnChange event generates the function which reads the steam table to find the required value, returns the result and writes it to an auxiliary parameter (A1) of the point.

Scripts

The OnAlarm script for the PV of the point, PressureCooker:

```
Sub PressureCooker_PV_OnChange()
    Dim temp, Pressure

    Pressure = ParamValue("pressureD1.PV")
    temp = ParamValue("tempD1.PV")
    ParamValue(".A1") = steamVolume(temp, pressure)
End Sub
```

The Library script (note that the Function and End Function statements are required):

```
Function steamVolume(temp, Pressure)
    dim file, filename, msgstream, tempdata

    Set file = CreateObject("Scripting.FileSystemObject")
    filename= "c:\server\user\steamtables.txt"
    Set msgstream = file.OpenTextFile(filename, ForReading, FALSE)
    ' Open for reading, do not create if non existent.
    ' skip first line with column labels
    Msgstream.SkipLine

    do
        msg = msgstream.ReadLine
        ' extract the value for the temperature
        readtemp = CSng(mid(msg,1,6))
        loop while readtemp < temp

        tempdata = CSng(mid(msg,13,6))

        do while tempdata < pressure
            msg = msgstream.ReadLine
            ' extract the value for the pressure
            tempdata = CSng(mid(msg,13,6))
        loop

        steamVolume = CSng(mid(msg,22,6))

        file.close
    End Function
```

Remarks

- The steam table must be in the format shown in the following example. It is ordered from lowest temperature to highest temperature, and rows with the same temperature are ordered from lowest pressure to highest pressure. The first line of the table contains the labels of the columns.
- This script uses the file path *c:\server\user* to write the *steamtables.txt* file, but you can use any convenient file path. It is only printed here as an example.

```
Temperature Pressure Specific volume
110          1.9       1.01
```


110	2.0	1.05
111	1.6	1.01

Related topics

“Copying example Script Editor scripts” on page 105

Requesting a task when two points are enabled

Scenario

You only want a task to run if two particular points are on scan.

Solution

You create a periodic script called 'ComparePoints' that checks the status of two points, 'Pump1' and 'Well1.' You set the script's period to an appropriate value, for example five minutes.

If both points are on-scan, the task is requested and the required parameters are sent to the task.

Scripts

```
Sub ComparePoints_OnTimer()  
    Dim PrmBlk  
  
    If (Server.ParamValue("Pump1.OnScan") AND Server.ParamValue("Well1.OnScan"))  
        Then  
            Set PrmBlk = Server.CreateParamBlock  
            PrmBlk.Param1 = 1  
            Server.RequestTask 111,PrmBlk  
        End If  
    End Sub
```

Remarks

- In addition to the period, you must define the periodic script's other properties.

Related topics

“Copying example Script Editor scripts” on page 105

Sending a report to a particular Station

Scenario

You want to call up a report on a particular Station each time it runs.

Solution

The report's OnCompletion event uses the Server Display program (LRN 21) to call up the report on the specified Station. (The Server Display Program performs numerous tasks, depending on the parameter values. For details about the Server Display program (LRN 21), see the *Server and Client Configuration Guide*.)

You create a parameter block so that you can pass the required parameters to LRN 21. In this scenario, you must specify the following.

ParamBlock Property (LRN Parameter)	Description
<i>crt</i>	The number of the Station on which the report is called up.
<i>Param1</i>	Set to <i>10</i> . (This instructs LRN 21 to call up the report specified in Param2 on the Station specified in crt.)
<i>Param2</i>	The report number.

Scripts

```
Sub Report11_OnCompletion()
    Dim ParamBlock

    Set ParamBlock = Server.CreateParamBlock

    ParamBlock.crt = 6
    ParamBlock.param1 = 10
    ParamBlock.param2 = 11

    Server.RequestTask 21, ParamBlock
End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Sending data to other displays

Scenario

You want to display a text message on a large wall-mounted display in response to certain events or conditions. The options for communicating with this display are:

- OPC
- DDE interface
- ASCII file via a serial link
- e-mail

Solution

Possible solutions are:

- OPC servers can be controlled via an automation interface. This can be accessed easily from VBScript.
- DDE function calls are accessible from a VB application, which would use `createobject("HSCAutomationServer.Server")` to create an instance of the server.
- Create a COM wrapper to DDE that would make DDE accessible from VB script.
- Write to an ASCII file then talk to a serial port via a communications control (such as MSComm).

Related topics

“Copying example Script Editor scripts” on page 105

Setting a point to a specific value at the start of each day

Scenario

At the start of each day, you want to set a point to a predetermined value. You also want the operation to be logged in the server's event log.

Solution

The way in which the set of controls should be started depends on the event used to start them. In the case of the OnDayStart event, this could be used to run a script of controls to be performed. There are two ways the journaling can be achieved:

- Turn on the **Journal parameter changes** option in the script engine.

Although this is the simpler option, it applies to all script-derived parameter changes.

- Create a library script that performs the control and logs the event.

This solution is more efficient because it only journals controls of interest. The library script would take the point name and the value as arguments.

Scripts

The Library script (note that the Sub and End Sub statements are required):

```
Sub ControlFunction(Server, name, Value)
    Dim AlMDtls

    Set AlMDtls = Server.CreateAlarmDetails
    AlMDtls.Description = "Point " & name & " controlled"
    AlMDtls.Value = Value
    AlMDtls.Priority = hscJournal
    AlMDtls.Area = ""

    Server.ParamValue(name & ".SP") = Value
    Server.GenerateAlarm AlMDtls
End Sub
```

The OnDayStart script for the Server:

```
Sub Server_OnDayStart()
    ControlFunction Server, "01IFBLUE", 36.7
End Sub
```

Related topics

“Configuring and managing script engines” on page 61

“Copying example Script Editor scripts” on page 105

Silencing a siren without acknowledging the alarm

Scenario

Operators want to be able to silence the siren without acknowledging any alarms.

Solution

You use two status points and a periodic script.

The point called 'SIREN' controls the siren, and the operators use the point called 'SILENCE' to disable the siren. (One way of giving operators control of the SILENCE point is to add a 'Silence' button to a display.)

The periodic script checks whether there are any unacknowledged alarms for asset 'AB.' If there are any unacknowledged alarms, the value of SILENCE is read to determine whether the siren should be disabled.

Scripts

You create a periodic script called 'SirenCheck' and give it an appropriate period. The script is as follows:

```
Sub SirenCheck_OnTimer()
  If (Server.Areas("AB").AlarmCount(hscAlarmCountAllUnack)
    > 0) Then
    If (StrComp(ParamValue("SILENCE.SP"), "SILENCE", vbTextCompare) = 0)
      Then
        Server.ParamValue("SIREN.SP") = "OFF"
      Else
        Server.ParamValue("SIREN.SP") = "ON"
      End If
    Else
      Server.ParamValue("SIREN.SP") = "OFF"
    End If
  End Sub
```

Related topics

“Copying example Script Editor scripts” on page 105

Using an auxiliary parameter to store calculated data

Scenario

You have a process variable, and want to calculate and present the RMS (Root Mean Square) value of that variable as a point parameter.

Solution

The OnChange event for the PV of the point (01IFBLUE) is used to run a script to calculate the RMS value and store it into another parameter (A1).

Scripts

The OnChange script for the PV of the point, 01IFBLUE:

```
Sub 01IFBLUE_PV_OnChange()  
    Dim RMS  
  
    RMS = ParamValue(".A1")  
    RMS = (RMS * RMS + EventInfo.value * EventInfo.value) / 2  
    ParamValue(".A1") = Sqr(RMS)  
End Sub
```

Remarks

- Parameter A1 must be initialized for the script above to yield correct values. Initialization can be done at the Server start, or in another script attached to the OnDayStart event of the server, or any other appropriate script or application.

Related topics

“Copying example Script Editor scripts” on page 105

Using auxiliary parameters to store and display data

Scenario

You want to calculate and present the running average value of a variable as a point parameter.

Solution

The OnTimer event for the point 01IFBLUE is used to run a script to calculate the running average value and store it in an auxiliary parameter (A1). The total number of samples is stored in another auxiliary parameter (A2).

Scripts

The OnTimer script for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnTimer()
    Dim Average, Samples

    Average = ParamValue(".A1")
    Samples = ParamValue(".A2")

    If (Samples < 100) Then
        ParamValue(".A1") = (Average * Samples + ParamValue(".PV")) / (Samples + 1)
        ParamValue(".A2") = Samples + 1
    Else
        ParamValue(".A1") = (Average * (Samples - 1) + ParamValue(".PV")) / Samples
    End If
End Sub
```

Remarks

- Parameters A1 and A2 must be initialized to yield correct values. Initialization can be done at the system start, or in another script attached to the OnDayStart event of the server, or any other appropriate script or application.

Related topics

“Copying example Script Editor scripts” on page 105

Working with server redundancy

Making sure a script's task can still be completed if a failover occurs requires extra consideration. This example, based on “Performing calculations on a process variable” on page 124, shows the changes that are required to cope with redundancy.

Two things need to be catered for:

- Loss of the OnDayStart event
- Creation of the timer if a failover occurs

Note that point 01IFBLUE needs to have the parameters 'Timer' and 'NextDayStart' created before these scripts are run.

Unaltered script code (that is, for a non-redundant server)

The OnDayStart script for the server:

```
Sub Server_OnDayStart()
    Funct Server
End Sub
```

The OnStartup script for the server:

```
Sub Server_OnStartup()
    Points("01IFBLUE").CreateTimer 2,3600,0
End Sub
```

The OnTimer script for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnTimer()
    Dim Average, Samples

    If (EventInfo.TimerID = 2) Then
        Average = ParamValue(".A1")
        Samples = ParamValue(".A2")

        ParamValue(".A1") = (Average * Samples + ParamValue(".PV")) / (Samples + 1)
        ParamValue(".A2") = Samples + 1
    End If
End Sub
```

The Library script (note that the Sub and End Sub statements are required):

```
Sub Funct(MyServer)
    MyServer.ParamValue("01IFBLUE.A1") = 50.0
    MyServer.ParamValue("01IFBLUE.A2") = 0
End Sub
```

Altered script code (modified for a redundant server)

The following code includes the additions required to capture the loss of the OnDayStart event.

Note that the extra code is shown in bold type.

The CommitValue function is used to immediately send the parameter value to the backup server.

The extra code stores the expected occurrence of the next OnDayStart event. If there is a failover to the backup, it causes the backup server to receive an OnStartup event. In the script code attached to this event, a check can be performed whether the next OnDayStart event is yet to occur. If it is past the expected occurrence time of the event, it performs the action that would have occurred on that event and stores the expected time of the next occurrence. Note that the next time it is stored as part of the usual behavior of the OnDayStart script.

The OnDayStart script for the server:

```
Sub Server_OnDayStart()
    Funct Server
End Sub
```

The OnStartup script for the server:

```
Sub Server_OnStartup()
    Dim TimeNow
    Dim NextStart

    TimeNow = Time
    NextStart = ParamValue("01IFBLUE.NextDayStart")

    If (NextStart < TimeNow) Then
        Funct Server
    End If

    Points("01IFBLUE").CreateTimer 2,3600,0
End Sub
```

The OnTimer for the point, 01IFBLUE:

```
Sub 01IFBLUE_OnTimer()
    Dim Average, Samples

    If (EventInfo.TimerID = 2) Then
        Average = ParamValue(".A1")
        Samples = ParamValue(".A2")

        ParamValue(".A1") = (Average * Samples + ParamValue(".PV")) / (Samples + 1)
        ParamValue(".A2") = Samples + 1
        CommitValue Array(".A1", ".A2")
    End If
End Sub
```

The Library script (note that the Sub and End Sub statements are required):

```
Sub Funct(MyServer)
    MyServer.ParamValue("01IFBLUE.A1") = 50.0
    MyServer.ParamValue("01IFBLUE.A2") = 0
    MyServer.ParamValue("01IFBLUE.NextDayStart") = NextStart + #24:00:00#
    MyServer.CommitValue Array("01IFBLUE.A1", "01IFBLUE.A2", "01IFBLUE.NextDayStart")
End Sub
```

The additions to the original code to counteract the loss of the OnTimer event are shown below.

As with correcting for the loss of the OnDayStart event, correcting for the loss of OnTimer requires checking the status of the timer when the OnStartup event occurs on the backup.

A timer that was running on the primary will not be automatically recreated on the backup. It can be restarted manually by storing the expected time of the next OnTimer event each time it occurs. The backup can then recreate the timer and pass the expected next time as the parameter specifying when the timer is to first occur.

In the OnStartup script, if the expected next time is set to an initial value of 0 then the timer is created. If there is a proper OnShutdown event, the expected next time is set back to 0.

The OnStartup event is normally used to initialize the data for the scripts so care must be taken that the code to cope with redundancy will not effect the normal initialization of the scripts when the server starts.

The OnDayStart script for the server:

```
Sub Server_OnDayStart()
    Funct Server
End Sub
```

The OnStartup script for the server:

```
Sub Server_OnStartup()
    Dim TimeNow
    Dim NextTime

    TimeNow = Time
    NextTime = ParamValue("01IFBLUE.Timer")

    If (NextTime = 0) Then
        CreateTimer 1,3600,0
        ParamValue("01IFBLUE.Timer") = TimeNow + #1:00:00#
        CommitValue "01BLUE.Timer"
    Else

```

```

        Points("01IFBLUE").CreateTimer 2,3600,NextTime
    End If
End Sub

```

The OnShutdown script for the server:

```

Sub Server_OnShutdown()
    ParamValue("01IFBLUE.Timer") = 0
    CommitValue "01IFBLUE.Timer"
End Sub

```

The OnTimer script for the point, 01IFBLUE:

```

Sub 01IFBLUE_OnTimer()
    Dim Average, Samples
    Dim TimeNow

    If (EventInfo.TimerID = 2) Then
        ParamValue("01IFBLUE.Timer") = Time() + #1:00:00#
        CommitValue "01IFBLUE.Timer"

        Average = ParamValue(".A1")
        Samples = ParamValue(".A2")

        ParamValue(".A1") = (Average * Samples + ParamValue(".PV")) / (Samples + 1)
        ParamValue(".A2") = Samples + 1
        CommitValue Array(".A1", ".A2")
    End If
End Sub

```

The Library script (note that the Sub and End Sub statements are required):

```

Sub Funct(MyServer)
    MyServer.ParamValue("01IFBLUE.A1") = 50.0
    MyServer.ParamValue("01IFBLUE.A2") = 0
    MyServer.CommitValue Array("01IFBLUE.A1", "01IFBLUE.A2")
End Sub

```

The two separate modifications can be combined to allow the script to cope with both events. The resulting code is shown below.

The OnDayStart script for the server:

```

Sub Server_OnDayStart()
    Funct Server
End Sub

```

The OnStartup script for the server:

```

Sub Server_OnStartup()
    Dim TimeNow
    Dim NextTime
    Dim NextStart

    TimeNow = Time
    NextTime = ParamValue("01IFBLUE.Timer")
    NextStart = ParamValue("01IFBLUE.NextDayStart")

    If (NextStart < TimeNow) Then
        Funct Server
    End If

    If (NextTime = 0) Then
        CreateTimer 1,3600,0
        ParamValue("01IFBLUE.Timer") = TimeNow + #1:00:00#
        CommitValue "01IFBLUE.Timer"
    Else
        Points("01IFBLUE").CreateTimer 1,3600,NextTime
    End If
End Sub

```

The OnShutdown script for the server:

```

Sub Server_OnShutdown()
    ParamValue("01IFBLUE.Timer") = 0

```

```

    CommitValue "01IFBLUE.Timer"
End Sub

```

The OnTimer script for the point, 01IFBLUE:

```

Sub 01IFBLUE_OnTimer()
    Dim Average, Samples
    Dim TimeNow

    If (EventInfo.TimerID = 1) Then
        ParamValue("01IFBLUE.Timer") = Time + #1:00:00#
        CommitValue "01IFBLUE.Timer"

        Average = ParamValue(".A1")    Samples = ParamValue(".A2")
        ParamValue(".A1") = (Average * Samples + ParamValue(".PV")) / (Samples + 1)
        ParamValue(".A2") = Samples + 1
        CommitValue Array(".A1", ".A2")
    End If
End Sub

```

The Library script (note that the Sub and End Sub statements are required):

```

Sub MySub(MyServer)
    MyServer.ParamValue("01IFBLUE.A1") = 50.0
    MyServer.ParamValue("01IFBLUE.A2") = 0
    MyServer.ParamValue("01IFBLUE.NextDayStart") = NextStart + #24:00:00#
    MyServer.CommitValue Array("01IFBLUE.A1", "01IFBLUE.A2", "01IFBLUE.NextDayStart")
End Sub

```

Related topics

“Script transfer in a redundant server system” on page 18

“Copying example Script Editor scripts” on page 105

Notices

Trademarks

Experion®, PlantScape®, SafeBrowse®, TotalPlant®, and TDC 3000® are registered trademarks of Honeywell International, Inc.

OneWireless™ is a trademark of Honeywell International, Inc.

Other trademarks

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Trademarks that appear in this document are used only to the benefit of the trademark owner, with no intention of trademark infringement.

Third-party licenses

This product may contain or be derived from materials, including software, of third parties. The third party materials may be subject to licenses, notices, restrictions and obligations imposed by the licensor. The licenses, notices, restrictions and obligations, if any, may be found in the materials accompanying the product, in the documents or files accompanying such third party materials, in a file named third_party_licenses on the media containing the product, or at <http://www.honeywell.com/ps/thirdpartylicenses>.

Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

<http://www.honeywellprocess.com/support>

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

hpsdocs@honeywell.com

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

<https://honeywell.com/pages/vulnerabilityreporting.aspx>

Submit the requested information to Honeywell using one of the following methods:

- Send an email to security@honeywell.com.
- or
- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, <https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx>.

Training classes

Honeywell holds technical training classes on Experion PKS. These classes are taught by experts in the field of process control systems. For more information about these classes, contact your Honeywell representative, or see <http://www.automationcollege.com>.

Index

A

- AlarmDetails object 70
- alarms
 - AlarmDetails object 70
 - CreateAlarmDetails method 86
 - creating 22
 - disabling 117
 - enabling 117
 - GenerateAlarm method 89
 - raising 22
- Alert object 71
- alerts
 - Alert object 71
 - Alerts collection 72
 - described 35
 - writing scripts for 35
- Alerts collection 72
- APIs
 - Server
 - custom applications 29
- applications
 - custom
 - security considerations 29
 - using the Server Scripting Object model 59
 - custom applications 29
 - external, launching/interacting with 27
- Area object 72
- Areas collection 73
- automatic script engines 56

C

- checking a script's progress 45, 48
- checkpointing, described 28
- collections
 - Alerts 72
 - Areas 73
 - Params 79
 - Points 80
 - Reports 82
 - Server Scripts 17
 - using in Server Scripts 19
- colors
 - coding of scripts 39
- CommitValue method 85
- comparing strings 23
- components
 - script 13
- Control Builder, using the Script Editor 32
- copying
 - example scripts 105
- CreateAlarmDetails method 86

- CreateParamBlock method 87
- CreateTimer method 87
- creating
 - alarms 22
- custom applications
 - security considerations 29

D

- data
 - storage in Server Scripting 21
- data storage 70, 78
- dates
 - specifying in Server Scripts 20
- debugging
 - error handlers 25
 - overview 43
 - scripts 43, 56
 - server scripts 66
- disabling
 - alarms 117
- display scripts, described 26
- draft scripts 33

E

- enabling
 - alarms 117
 - script engines
 - automatic 62
 - manual 64
- engines, script 55
- enumerations
 - point parameters 24
 - Server Script collections 17
- error handling
 - server scripting error handlers 25
- error messages
 - server scripting 50
- EventInfo object 74
- events
 - OnAcknowledge 96
 - OnAlarm 96
 - OnAlert 97
 - OnAlertAcknowledge 97
 - OnAlertNormal 98
 - OnChange 98
 - OnCompletion 99
 - OnDayStart 99
 - OnNormal 99
 - OnOperChange 100

- OnShutDown 100
- OnStartup 101
- OnTimer 101
- Server Scripts 14
- simulating 44
- examples
 - copying example scripts 105
 - scenarios 103
 - scripts 103, 107–111, 113, 115–128, 130–136
- execution of server scripts 12

F

- field addresses 28
- FireScriptEvent method 88
- flexible points
 - described 34
 - scripts 34
 - storing script values with 21

G

- GenerateAlarm method 89
- GenerateMessage method 90
- global variables in Server Scripting 21

H

- help for VBScript 13

I

- iterative tasks in Server Scripts 17

J

- journaling changes in parameter values 62, 64

K

- keyboards
 - shortcuts 40
- KillTimer method 91

L

- languages
 - scripting 13
- launching external applications 27
- library scripts
 - described 11
 - writing 36
- log, server 49
- LogMessage method 92

M

- manual script engines 56
- messages

- error 50
- methods
 - CommitValue 85
 - CreateAlarmDetails 86
 - CreateParamBlock 87
 - CreateTimer 87
 - FireScriptEvent 88
 - GenerateAlarm 89
 - GenerateMessage 90
 - KillTimer 91
 - LogMessage 92
 - ModifyTimer 92
 - Request 93
 - RequestTask 93
 - SendFileToBackup 94
 - TimerExists 95
- Models
 - Server Scripting Object Model 68
- ModifyTimer method 92

O

- objects
 - AlarmDetails 70
 - Alert 71
 - Alerts collection 72
 - Area 72
 - Areas collection 73
 - EventInfo 74
 - Param 77
 - ParamBlock 78
 - Params collection 79
 - Point 79
 - Points collection 80
 - referencing in Server Scripts 19
 - Report 81
 - Reports collection 82
 - Script 82
 - Server 83
 - Server Scripting Model 68
- OnAcknowledge event 96
- OnAlarm event 96
- OnAlert event 97
- OnAlertAcknowledge event 97
- OnAlertNormal event 98
- OnChange event 98
- OnCompletion event 99
- OnDayStart event 99
- online scripts 33
- OnNormal event 99
- OnOperChange event 100
- OnShutdown 59
- OnShutdown event 100
- OnStartup 59
- OnStartup event 101
- OnTimer event 101
- operator interaction
 - external applications 27
 - scripts 26

P

- Param object 77
- ParamBlock object 78
- parameters
 - Param object 77
 - point parameters
 - enumerated 24
- Params collection 79
- periodic scripts
 - described 10
 - operation 10
 - properties 36
 - writing 36
- Point object 79
- points
 - enumerated point parameters 24
 - flexible
 - described 34
 - storing script values with 21
 - parameters
 - enumerated 24
 - process
 - described 32
 - scripts, writing 32
 - storing script values with 21
 - standard
 - described 33
 - scripts, writing 33
 - storing script values with 21
 - user-defined parameters 21
 - storing script data 21
 - timers 15
- Points collection 80
- process points, scripts for 32
- projects
 - Quick Builder 33

Q

- queues, script 57
- Quick Builder
 - projects 33
 - Script Editor 33

R

- raising alarms 22
- redundant-server systems
 - example scripts 137
 - issues 28
 - script transfer 18
 - sending files to the backup 94
 - writing changes to the backup 85
- reenabling
 - script engines 63, 65
- referencing objects in Server Scripts 19
- Report object 81
- reports
 - scripts 109, 119, 120, 131

- described 10
 - writing 38
- timers 15
- Reports collection 82
- Request method 93
- RequestTask method 93
- run-time error messages 50

S

- scenarios (examples)
 - alarms 113, 117, 118, 126, 127, 134
 - calculations 124
 - data 108, 128, 132
 - displays 107
 - events 122
 - launching applications 123
 - messages 121
 - points 110, 111, 113, 115–117, 125, 127, 130, 133, 135, 136
 - reports 109, 119, 120, 131
 - scripting scenarios 103
- Script Editor
 - in Control Builder 32
 - in Quick Builder 33
 - using 31, 39, 40
- script engines
 - assignment of scripts 56
 - automatic 56
 - enabling 62
 - reenabling 63
 - configuring and managing 61
 - debugging 66
 - described 55
 - manual 56
 - enabling 64
 - reenabling 65
 - queues 57
 - time-outs 58
- Script object 82
- scripting
 - components 13
 - execution 12
 - library 10, 11
 - periodic 10
 - point 10
 - report 10
 - scripting languages 13
 - server 10
 - Server Scripting Object Model 68
 - size (lines of code) 12
 - time-outs 58
 - types 10
- scripts
 - alerts 35
 - assignment to script engines 56
 - basics 9
 - color-coding 39
 - copying examples 105
 - debugging 43, 66

- display 26
 - draft 33
 - editor 31
 - enabling
 - automatic script engines 62
 - manual script engines 64
 - engines 55
 - examples 103, 107–111, 113, 115–128, 130–136
 - interaction
 - external applications 27
 - operator 26
 - online 33
 - periodic 36
 - queues 57
 - scenarios 103, 107–111, 113, 115–128, 130–136
 - script engines
 - automatic, reenabling 63
 - configuring and managing 61
 - manual, reenabling 65
 - Server Scripting Object Model 68
 - storing script-derived data 21
 - testing 43
 - security
 - custom applications 29
 - SendFileToBackup method 94
 - Server object 83
 - server scripting
 - basics 9
 - collections 17
 - data storage 21
 - debugging 66
 - error handlers 25
 - events 14
 - examples 107–111, 113, 115–128, 130–136
 - execution 12
 - languages 13
 - library 11
 - periodic 10
 - script engines
 - automatic, enabling 62
 - automatic, reenabling 63
 - configuring and managing 61
 - manual, enabling 64
 - manual, reenabling 65
 - Server Scripting Object Model 68
 - size of, limitation 12
 - time/date specifying 20
 - timers 15
 - types 10
 - servers
 - API 29
 - log, interpreting 49
 - redundant system
 - example scripts 137
 - issues 28
 - script transfer 18
 - script engines
 - automatic, reenabling 63
 - automatic, status 62
 - configuring and managing 61
 - manual, reenabling 65
 - manual, reenabling 65
 - manual, status 64
 - scripts 9, 36
 - Server API 29
 - startup and shutdown 59
 - timers 15
 - shortcuts
 - keyboard 40
 - shutdown of server 59
 - simulating events 44
 - size of scripts (lines of code) 12
 - standard points 33
 - startup of server 59
 - Stations
 - display scripts 26
 - script engines
 - automatic, status 62
 - configuring and managing 61
 - manual, status 64
 - reenabling 63, 65
 - storing
 - data 70, 78
 - script-derived script data 21
 - strings, comparing 23
 - System Configuration
 - Script Engines
 - debugging 66
 - system interfaces
 - flexible points 21
- ## T
- tasks
 - custom applications 29
 - testing and debugging scripts
 - checking a script's progress 45, 48
 - overview 43
 - simulating events 44
 - using a dedicated script engine 56
 - time-outs
 - script engine time-outs 58
 - TimerExists method 95
 - timers
 - CreateTimer method 87
 - KillTimer method 91
 - ModifyTimer method 92
 - OnTimer event 101
 - periodic scripts 10
 - Server Scripts 15
 - time/date, specifying in Server Scripts 20
 - TimerExists method 95
 - times
 - specifying in Server Scripts 20
 - transfer of scripts in a redundant-server system 18
 - troubleshooting
 - error messages 50
 - script engine time-outs 58
 - scripting errors 50
 - server log 49
 - server scripting error messages 50

U

- user interaction with scripts 26
- user-defined parameters
 - storing script-derived data 21
- utilities
 - custom applications 29

V

- variables
 - global, in Server Scripting 21
- VBScript
 - error handlers 25
 - Server Scripting 13

