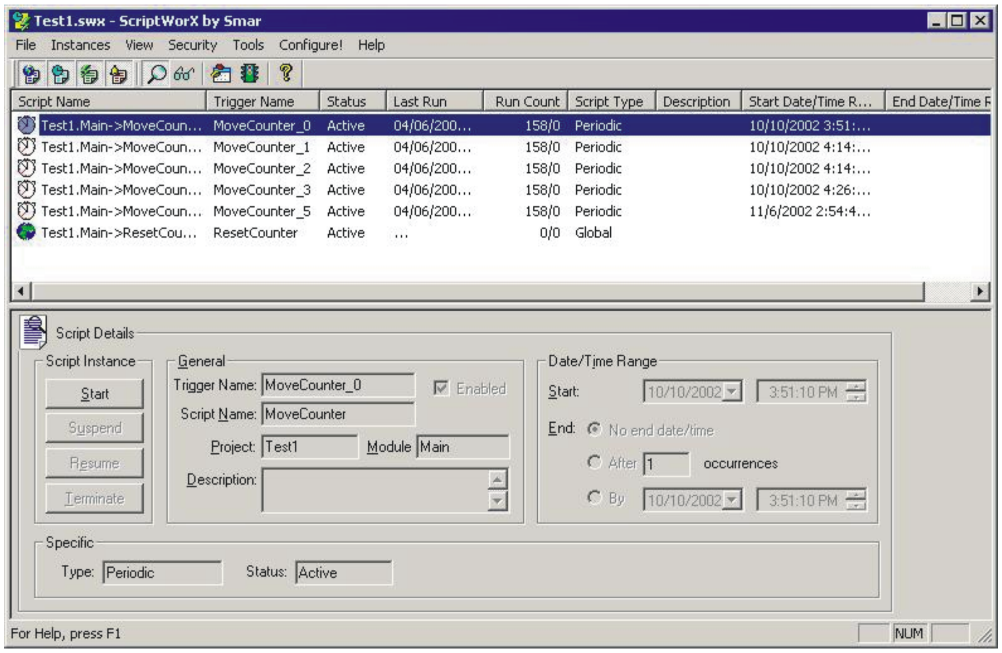
ScriptWorX

USER`S GUIDE

ScriptWorX


Test1.swx - ScriptWorX by Smar
File Instances View Security Tools Configure! Help
For Help, press F1
NUM

Script Name	Trigger Name	Status	Last Run	Run Count	Script Type	Description	Start Date/Time R...	End Date/Time R
Test1.Main->MoveCoun...	MoveCounter_0	Active	04/06/200...	158/0	Periodic		10/10/2002 3:51:...	
Test1.Main->MoveCoun...	MoveCounter_1	Active	04/06/200...	158/0	Periodic		10/10/2002 4:14:...	
Test1.Main->MoveCoun...	MoveCounter_2	Active	04/06/200...	158/0	Periodic		10/10/2002 4:14:...	
Test1.Main->MoveCoun...	MoveCounter_3	Active	04/06/200...	158/0	Periodic		10/10/2002 4:26:...	
Test1.Main->MoveCoun...	MoveCounter_5	Active	04/06/200...	158/0	Periodic		11/6/2002 2:54:4...	
Test1.Main->ResetCou...	ResetCounter	Active	...	0/0	Global			

Script Details

Script Instance: Start, Suspend, Resume, Terminate

General
Trigger Name: MoveCounter_0 ☑ Enabled
Script Name: MoveCounter
Project: Test1 Module: Main
Description:

Date/Time Range
Start: 10/10/2002 3:51:10 PM
End: ● No end date/time
○ After 1 occurrences
○ By 10/10/2002 3:51:10 PM

Specific
Type: Periodic Status: Active

JUN / 04

ScriptWorX

VERSION 7.1

PVI EWS WK ME

Index

ScriptWorX

Introduction

ScriptWorX is the latest in the suite of automation products offered by Smar. The main function of ScriptWorX is to manage and maintain scripts that perform different functions. It takes advantage of multithreading in Visual Basic. The use of multithreading enables operating systems to run more efficiently because it assigns different tasks to different threads so that the CPU usage will be divided among these threads. This is very useful if you have a need or desire to run independent tasks at the same time.

For example, GraphWorX makes it is necessary to enter a sleep function in the code if you want or need two different tasks to run "at the same time". We say "at the same time" because in actuality the tasks are not running at the same time; they are running for short periods of time, freezing and then running again when it is their "turn" to run again. In essence, this is what multithreading accomplishes, but it is not necessary to enter any additional code.

ScriptWorX Architecture

ScriptWorX is currently supported on Windows 98, 2000, XP, and Windows NT. ScriptWorX is made up of three different components:

1. The server (ScriptWorX Engine)
2. The ScriptWorX user interface
3. The Visual Basic Editor

ScriptWorX Engine

The Engine is an invisible server that manages threads on which the scripts run. It hosts Visual Basic for Applications (VBA) Multithreading and is started automatically by its clients. The server will launch, monitor, and control VBA scripts, and it will host the Periodic32, Event32 and Alarm servers. This server is written as an ATL Server (.exe) or as an NT service.

ScriptWorX User Interface

The ScriptWorX user interface is where you will configure script trigger conditions. It allows configuration of Periodic32, Event32, and Alarm servers. During runtime, the user interface allows you to monitor scripts without running them. On start it automatically launches the ScriptWorX Engine.

ScriptWorX VBA

As discussed above, the greatest advantage of VBA is the availability of multithreading. ScriptWorX VBA 6.3 allows for creating, editing, and debugging multithreaded VBA scripts.

Note
Each of these components will be discussed later in greater detail.

ScriptWorX Features

ScriptWorX allows you to create and run custom scripts. These scripts are written in Microsoft Visual Basic for Applications, and are run as multithreaded objects. The ScriptWorX user interface indicates what type of script is being created and how it will be executed.

- **Global scripts** are either manually launched from outside the application or upon entering runtime mode.
- **Periodic scripts** are launched according to previously configured time restrictions.

- **Event scripts** are based on OPC expression changes.
- **Alarm scripts** are based on OPC alarms.

Additionally, ScriptWorX also implements the use of thread pool. Traditionally, when a script is run it starts a thread, which will be deleted by the system when it is done running the script. Each system is allowed to run a certain number of threads at a time due to the multithreading available in Visual Basic for Applications 6.3. With multithreading, the threads are not deleted at the end of a script instance but are instead reused by several miscellaneous scripts. The process of reusing threads will increase the performance of the scripts since they do not have to go through the process of creating the thread themselves.

ScriptWorX Engine

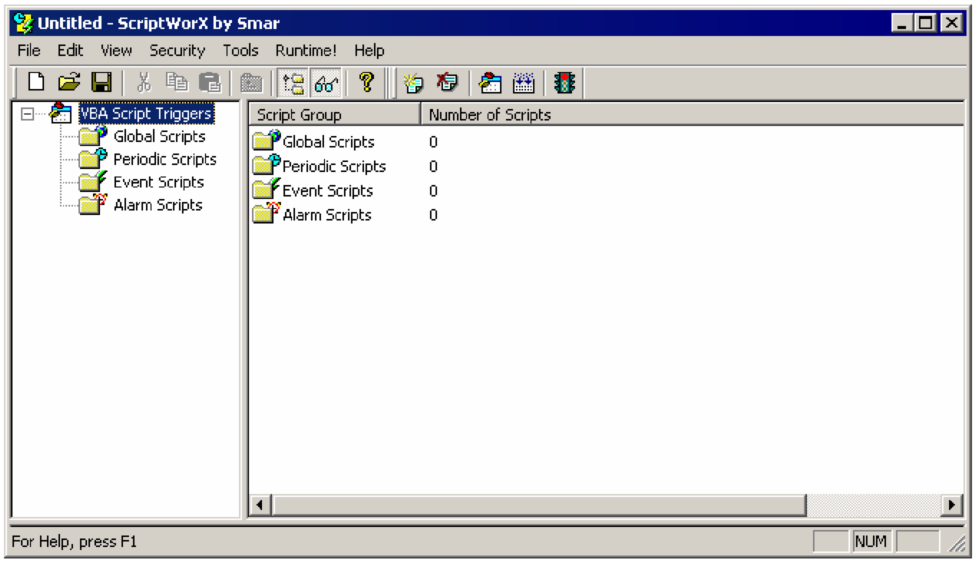
The ScriptWorX Engine is a server that communicates between the ScriptWorX user interface and the Visual Basic Editor. It creates scripts based on the configuration in the user interface.

The ScriptWorX Engine is started from ScriptWorX32.exe or from other clients, and is invisible to the user. It is an ATL application, which hosts the VBA instance and has the ability to run and manage script instances. Inside ScriptWorX32.exe also sits the Periodic32, Event32, and Alarm servers. These servers can be programmed (through automation) from the ScriptWorX user interface or from other sources. When put into runtime mode, they can start scripts based on their internal/external events. They then call the scripting engine to run the desired script.

User Interface

To start ScriptWorX from the Windows **Start** menu, select **Programs > Smar > ProcessView > ScriptWorX.**

The ScriptWorX user interface, shown below, has an Explorer look and feel to it, which allows for easy navigation and organization of scripts. In the user interface, you will configure scripts that the ScriptWorX32.dll will send to the Visual Basic Editor to run. It is able to configure the Periodic, Event, and Alarm servers (sitting in ScriptWorX) and monitor the running script instances.



Script Group	Number of Scripts
Global Scripts	0
Periodic Scripts	0
Event Scripts	0
Alarm Scripts	0

Figure 1. ScriptWorX User Interface

Toolbars

ScriptWorX has three toolbars with various functions. For more information on these functions, please refer to the **Menus** section.

- Standard toolbar
- Data-manipulation Toolbar
- Runtime toolbar

Standard Toolbar

To display the **Standard** toolbar, shown below, select **Standard Buttons** from the **Toolbars** submenu on the **View** menu. The **Standard** toolbar contains the following command buttons.

New: Creates a new document.

Open: Opens an existing document.

Save: Saves the active document.

Cut: Deletes the current selection, sending it to the Clipboard.

Copy: Copies the current selection to the Clipboard.

Paste: Pastes the current contents of the Clipboard.

Up One Level: Moves up one level in the tree view.

Tree Pane: Shows/hides the tree view.

Output Console: Shows/hides the output console.

About: Displays information about the application.

Data-Manipulation Toolbar

To display the **Data-Manipulation** toolbar, shown below, select **Data Manipulation Buttons** from the **Toolbars** submenu on the **View** menu. The **Data-Manipulation** toolbar contains the following command buttons.

Insert Trigger: Inserts a new script trigger.

Delete Trigger: Deletes script trigger.

Visual Basic Editor: Opens the Visual Basic Editor.

Make VBA DLL: Makes the VBA MT DLL (global project only).

Runtime: Enters the display into runtime mode.

Runtime Toolbar

The **Runtime** toolbar, shown below, is available only during runtime mode.

Global Scripts: Shows global scripts.

Periodic Scripts: Shows periodic scripts.

Event Scripts: Shows event scripts.

Alarm Scripts: Shows alarm scripts.

Details: Shows/hides the Script Details at the bottom of the screen.

Output Console: Shows/hides the output console.

Visual Basic Editor: Opens the Visual Basic Editor.

Configure: Returns the display to configuration mode.

About: Displays information about the application.

Menus

The menu bar consists of the following menus:

- File Menu
- Edit Menu
- View Menu
- Security
- Tools Menu
- Runtime Menu
- Help Menu

File Menu

The **File** menu contains the following commands:

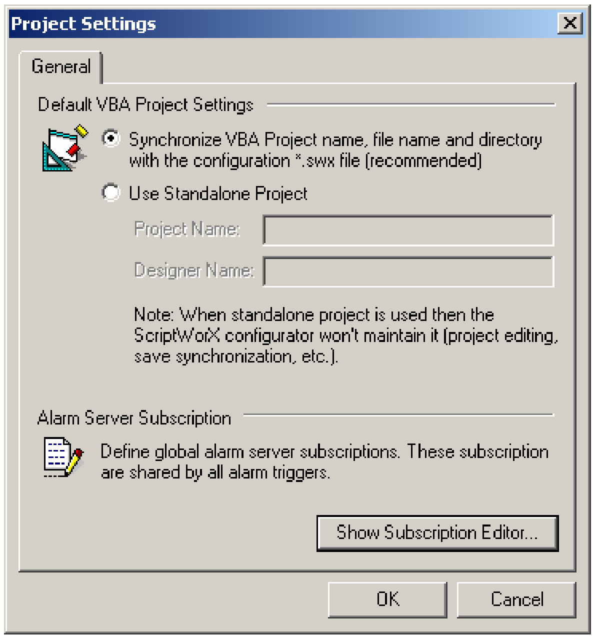
Command	Function
New (CTRL+N)	Opens a new ScriptWorX display (*.swx) file.
Open (CTRL+O)	Opens a dialog box that allows you to select an existing (*.swx) file to open.
Save (CTRL+S)	Saves the open *.swx display file. If this is the first time that the display is being saved, it will open the **Save As** dialog box.
Save As	Opens the **Save As** dialog box, which allows you to enter a file name for the ScriptWorX file. It also allows you to save the file under the same name, overwriting a previously saved *.swx file.
Settings (ALT+F7)	Opens the **Project Settings** dialog box, which allows you to edit the general VBA settings. For more information, please see the **Alarm Server Subscriptions** section.
Run VBA Project (CTRL+P)	Runs the global VBA project in Debug mode.
Make VBA DLL (CTRL+B)	Allows you to build the .dll file without actually being in VBA. This is helpful since it is not always desired to build the .dll directly after entering the code; you may wish to change the configuration of the script (i.e. what triggers it) after you have written the script.
Exit	Exits the application.

Project Settings

Selecting **Settings** from the **File** menu opens the **Project Settings** dialog box, which allows you to edit the default VBA project settings. You can choose to synchronize the VBA project name, file name, and directory with the configuration *swx file (recommended). Alternatively, you can use a standalone project and specify a **Project Name** and **Designer Name.**

Note
When a standalone project is used, the ScriptWorX Configurator will not maintain it.

You can also define global alarm server subscriptions, which are shared by all alarm triggers. For more information, please see the **Alarm Server Subscriptions** section.


Project Settings
General
Default VBA Project Settings
Project Name:
Designer Name:
Alarm Server Subscription
Show Subscription Editor...
OK
Cancel

[✓ Synchronize VBA Project name, file name and directory with the configuration *.swx file (recommended)]

Use Standalone Project

Note: When standalone project is used then the ScriptWorX configurator won't maintain it (project editing, save synchronization, etc.).

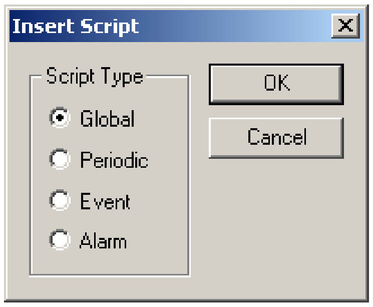
Define global alarm server subscriptions. These subscription are shared by all alarm triggers.

Figure 2. Project Settings Dialog Box

Edit Menu

The **Edit** menu contains the following commands:

Command	Function
Insert New Trigger (CTRL+T)	Opens the **Insert Script** dialog box, shown below, which allows you to insert a new Global, Periodic, Event, or Alarm script. For more information, refer to the **Different Types of Scripts** section.
Delete Trigger	Deletes the selected script trigger.



[✓ Global] Periodic Event Alarm

Figure 3. Insert Script Dialog Box

View Menu

The **View** menu contains the following commands:

Command	Function
Toolbars	Hides/shows the **Data-Manipulation** toolbar and the **Standard** toolbar.
Status Bar	Hides/shows the status bar.
Tree Pane	Displays the tree view in the left-hand pane, which allows for easy navigation through the different scripts and script types.
Output Console	Displays the output console (bottom pane) in the user interface.
Clear Output Console	Clears all output console events.
Output Console Widow Font	Opens the **Font** dialog box, which allows you to choose the text attributes for the output console.
Focus Tree View (ALT+1)	Places the cursor in the tree view (left-hand pane) of the interface.
Focus Right View (ALT+2)	Places the cursor in the right-hand pane of the interface.
Focus Console View (ALT+3)	Places the cursor in the output console.
Up One Level (CTRL+U)	Moves the cursor up one level.
Select Language (CTRL+ALT+U)	Opens the **Select Language** dialog box, shown below.

Select Language

The **Select Language** function on the **View** menu allows you to choose which language you want to use in your display. Choosing **Select Language** from the **View** menu opens the **Select Language** dialog box, shown below.

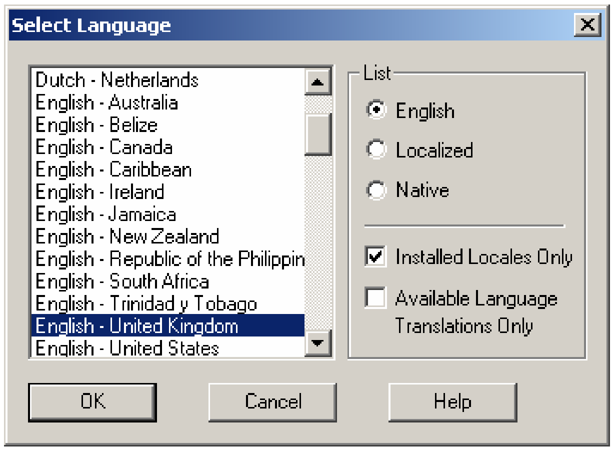


Figure 4. Select Language Dialog Box

Define the parameters listed in the table below. Then click **OK** to return to the work area.

Select Language Parameters

Parameter	Description
List	Lists available languages. Depending on which item you have selected, the view on the left will change. If **English** is checked, the languages will appear as their English name. If **Localized** is checked, the languages will appear with the native country in parentheses (for languages with several dialects only). When **Native** is checked, the languages are displayed the way they would be written in that language.
Installed Locales Only	If this is checked, local languages appear in the box.
Available Language Translations Only	Checking this box allows you to choose from available language translations only.

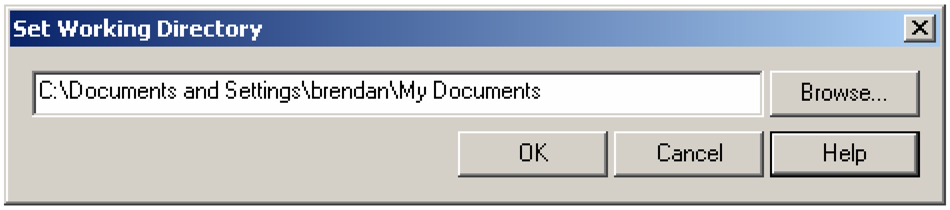
Security Menu

Selecting **Login** from the **Security** menu opens the Smar Security Login Utility. For more information, please refer to the Smar Security documentation.

Tools Menu

The **Tools** menu contains the following commands:

Command	Function
Macros	Launches either the Visual Basic Editor.
Security Configuration	Launches the Smar Security Configurator. For more information, please refer to the Smar Security documentation.
Options	Opens the **ScriptWorX Options** dialog box, which allows you to select general and event options. For more information, see below.
Set Working Directory	Opens the **Set Working Directory** dialog box, shown below, which allows you to select the directory where all files associated with the current scripts will reside. This is the directory to which the VBA files and dll files should be saved. Click **Browse** to select a directory.



C:\Documents and Settings\brendan\My Documents

Figure 5. Set Working Directory Dialog Box

Options

Selecting **Options** from the **Tools** menu opens the **ScriptWorX Options** dialog box, which contains two tabs: **General** and **Events**.

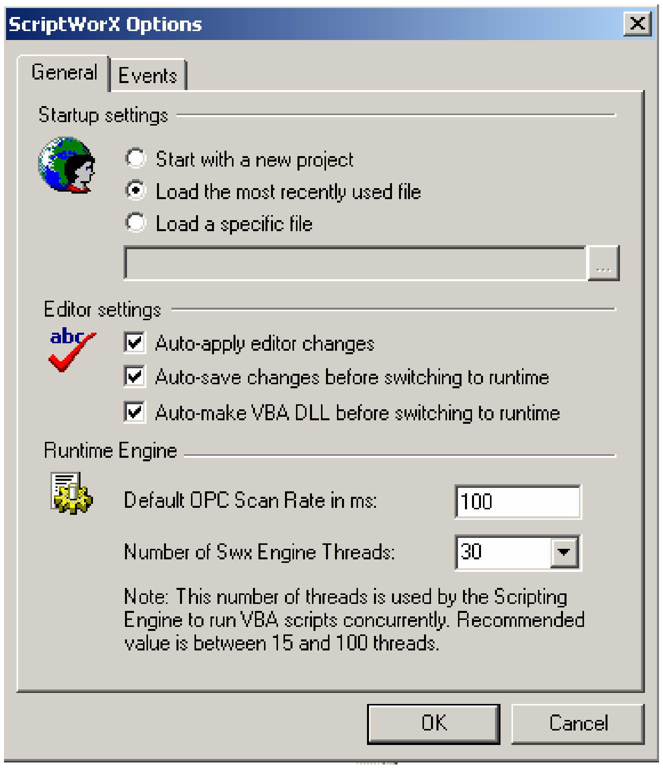
General Tab

The **General** tab, shown below, allows you general application settings. In the **Startup Settings** field, you can choose to start with a new project, to start with the most recently used file, or to start with a selected file. You can also choose the **Editor** settings. In the **Runtime Engine** section, you can specify the number of threads for the ScriptWorX engine.

The **Default OPC Scan Rate** (in milliseconds) is used as "default" OPC scan rate for tags, which does not override it using point extension syntax. The **Point Extension Syntax (PES)** allows for retrieving additional information related to OPC tags, such as quality and timestamp. The following are example expressions using a valid PES request:

- tag:Smar.Simulator\SimulatePLC.Ramp#timestamp
- tag:Smar.Simulator\SimulatePLC.Ramp#quality
- tag:\\pc1\Smar.Simulator\SimulatePLC.Ramp#timestamp
- tag:\\pc1\Smar.Simulator\SimulatePLC.Ramp#quality

Sometimes it may be necessary to enforce the "request data type" to a specific type, such as "string," in order to display this information in a process point.



Startup settings

- ( ) Start with a new project
- (●) Load the most recently used file
- ( ) Load a specific file

Editor settings

- [✓] Auto-apply editor changes
- [✓] Auto-save changes before switching to runtime
- [✓] Auto-make VBA DLL before switching to runtime

Runtime Engine

Default OPC Scan Rate in ms: 100

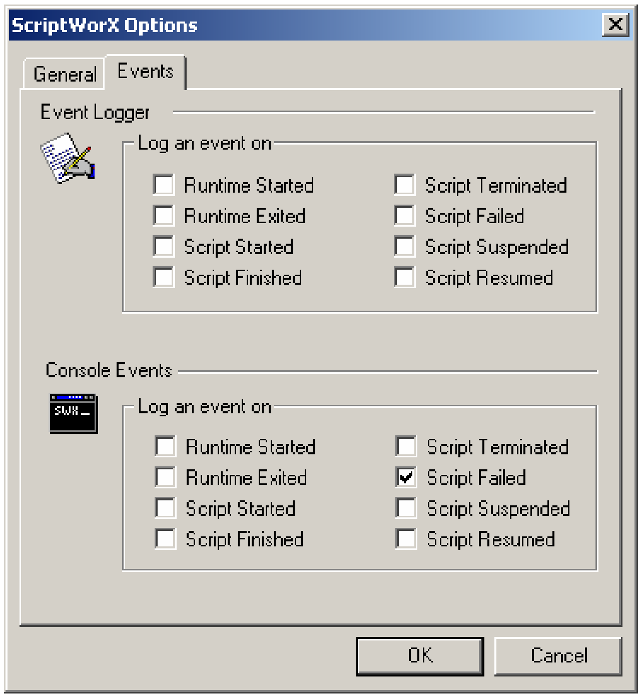
Number of Swx Engine Threads: 30

Note: This number of threads is used by the Scripting Engine to run VBA scripts concurrently. Recommended value is between 15 and 100 threads.

Figure 6. ScriptWorX Options Dialog Box: General Tab

Events Tab

The **Events** tab, shown below, allows you to select when you want to have events logged. You can choose all, some, or none of the options. Each option pertains to an event occurring within ScriptWorX.



Event Logger

Log an event on

- [ ] Runtime Started
- [ ] Runtime Exited
- [ ] Script Started
- [ ] Script Finished
- [ ] Script Terminated
- [ ] Script Failed
- [ ] Script Suspended
- [ ] Script Resumed

Console Events

Log an event on

- [ ] Runtime Started
- [ ] Runtime Exited
- [ ] Script Started
- [ ] Script Finished
- [ ] Script Terminated
- [✓] Script Failed
- [ ] Script Suspended
- [ ] Script Resumed

Figure 7. ScriptWorX Options Dialog Box: Events Tab

Runtime Menu

The **Runtime** menu toggles the display between runtime mode and configuration mode. When the display enters runtime mode, this menu will change to **Configure,** which returns the display to configuration mode. In addition to the **Runtime** toolbar, the following menus are also available during runtime mode.

- File
- Instances
- View
- Security
- Tools
- Help

File Menu

Command	Function
Exit	Quits the application and prompts to save documents.

Instances Menu

Command	Function
Start	Starts an instance of the selected script.
Suspend	Suspends an instance of the selected script.
Resume	Resumes an instance of the selected script.
Terminate	Terminates an instance of the selected script.

View Menu

Command	Function
Toolbar	Hides/shows the **Runtime** toolbar.
Status Bar	Hides/shows the status bar.
Details Pane	Hides/shows the Script Details at the bottom of the screen.
Output Console	Displays the output console (bottom pane) in the user interface.
Output Console Widow Font	Opens the **Font** dialog box, which allows you to choose the text attributes for the output console.
Show Global Scripts	Shows the global scripts in the top pane of the screen.
Show Alarm Scripts	Shows the alarm scripts in the top pane of the screen.
Show Event Scripts	Shows the event scripts in the top pane of the screen.
Show Periodic Scripts	Shows the periodic scripts in the top pane of the screen.
Focus Right View (ALT+2)	Places the cursor in the right-hand pane of the interface.
Focus Console View (ALT+3)	Places the cursor in the output console.
Select Language (CTRL+ALT+U)	Opens the **Select Language** dialog box.

Security Menu

Selecting **Login** from the **Security** menu opens the Smar Security Login Utility. For more information, please refer to the Smar Security documentation.

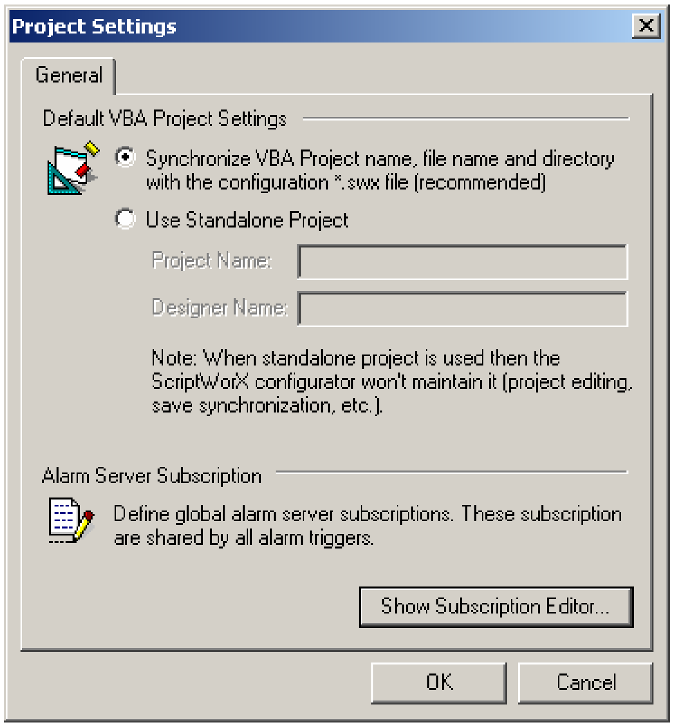
Help Menu

The **Help** menu contains the following commands:

Command	Function
Help Topics	Opens the help documentation associated with this program.
About Application	Opens the About Box, which provides the version number and copyright information for this software.

Alarm Server Subscriptions

The **Alarm Server Subscription** section in the **Project Settings** dialog box, shown below, determines what type of OPC connection will be made with the Current Events Viewer.


Project Settings
General
Default VBA Project Settings
Use Standalone Project
Project Name:
Designer Name:
Alarm Server Subscription
Show Subscription Editor...
OK
Cancel

(● Synchronize VBA Project name, file name and directory with the configuration *.swx file (recommended)

Note: When standalone project is used then the ScriptWorX configurator won't maintain it (project editing, save synchronization, etc.).

Define global alarm server subscriptions. These subscription are shared by all alarm triggers.

Figure 8. Project Settings Dialog Box

Clicking **Show Subscription Editor** opens the **Subscription Properties** dialog box, shown below, which allows you to add, edit, delete, and rename subscriptions. To add a new subscription, click **Add.** A subscription called "New Subscription" will appear as shown below.

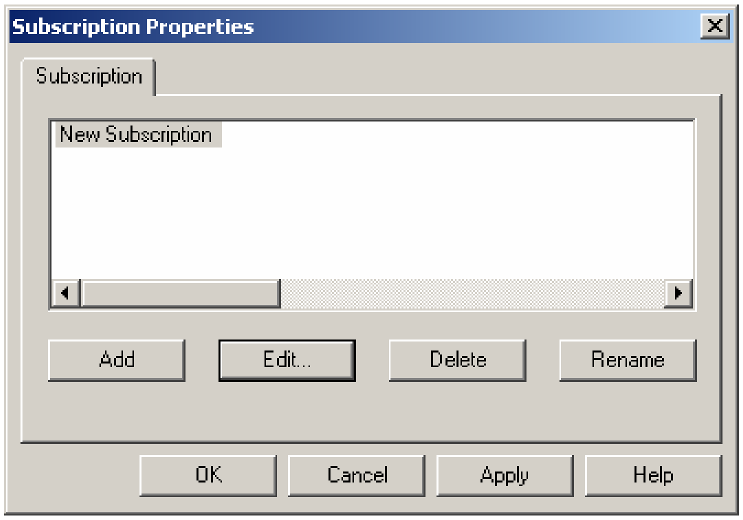


Figure 9. Subscription Properties Dialog Box

This subscription does not contain any data, so it is necessary to immediately edit the new subscription. To do so, click **Edit** to open the **Event Subscription** dialog box, which contains the following tabs.

- Server
- Types
- Categories
- Areas
- Source
- Attributes

Note
It is possible for a script to have more than one subscription. In fact, it is a very effective way to achieve filtering.

Server

The **Server** tab, shown below, allows you to select a **Node** and an **Event Server** for each subscription. To select the event server, click the **Browse.** This opens the OPC Universal Tag Browser, which allows you to choose from a list of available Alarm OPC servers. Select the desired server, and then click **OK.** For local servers, it is not necessary to fill in the **Node** field.

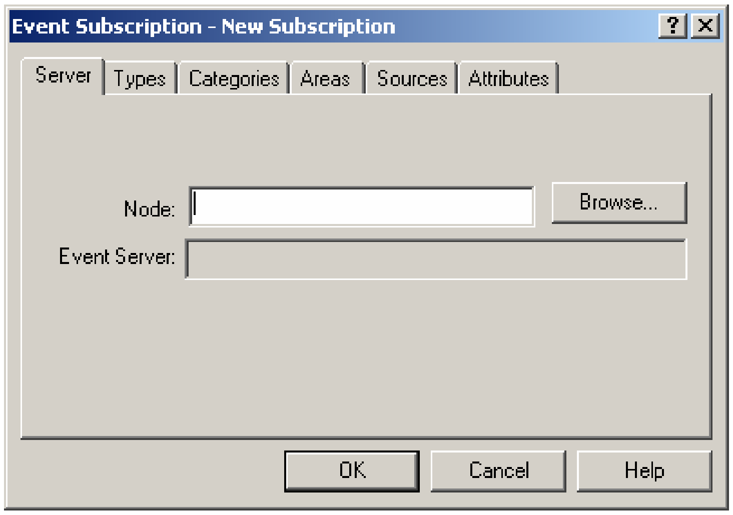
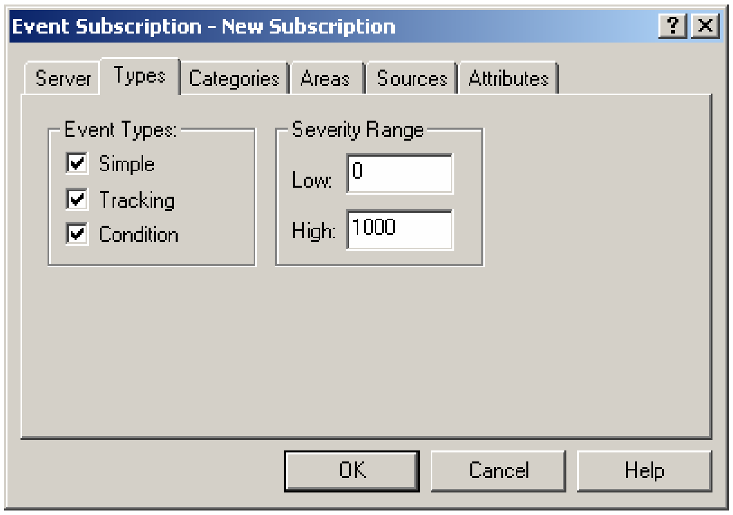


Figure 10. Alarm Subscription: Server Tab

Types

The **Types** tab, shown below, allows you to configure which OPC-defined event types each subscription should have, and it enables you to set the ranges for severity (priority). The value "0" is the low severity value, and "1000" is the high severity value. Please note that OPC Alarm and Events (AE) servers are required to scale severity values to the OPC ranges (i.e. an alarm and event server that contains two severity ranges will convert these to 0 and 1000).



Event Types:
- [✓] Simple
- [✓] Tracking
- [✓] Condition

Severity Range
Low: 0
High: 1000

Figure 11. Alarm Subscription: Types Tab

Event Types

Simple: These messages state information, but they do not have alarm status and do not contain information on what initiated the message. This includes the following information: Source, Time, Type, EventCategory, Severity, Message, Cookie, and Server-specific items.

Example: "FIC101, 12:0:0 1/1/99, Simple, Category1, 100, 'Shift Change', 1"

Simple messages would be similar to an event.

Tracking: These messages contain additional information about the client that initiated the event, including the following information: Source, Time, Type, EventCategory, Severity, Message, Cookie, ActorID, and Server-specific items.

Example: "FIC101, 12:0:1 1/1/99, Tracking, Category1, 300, 'Pump pressure Set to 10 psi', 1, Station 12"

Tracking messages are similar to event messages in that the cause of the event is important. An example would be an operator changing a setpoint value. This type of message does not include acknowledge capability.

Condition: These messages contain all of the above information but also include an acknowledgement portion. This includes the following information: Source, Time, Type, EventCategory, Severity, Message, Cookie, ConditionName, SubConditionName, ChangeMask, NewState, ConditionQuality, AckRequired, ActiveTime, ActorID, and Server-specific items.

Example: "FIC101, 12:0:3 1/1/99, Condition, Category1, 700, 'Pump pressure to high', 1, Limit, HiHi, 1,Active Enabled, Good, TRUE, 12:0:2 1/1/99"

Condition messages are considered "typical" alarm messages with acknowledge capability.

For exact details on any of the included information, please see the OPC Alarm and Events specification.

Categories

The **Categories** tab, shown below, allows you to select the categories for a subscription. Select the category from the list of available categories and click the **Add ->** button. The category will appear in the **Subscribed** list. To remove a category from this list, select the category in the **Subscribed** list and then click the **<- Remove** button. If no categories are listed in the **Subscribed** list, then all categories are selected by default.

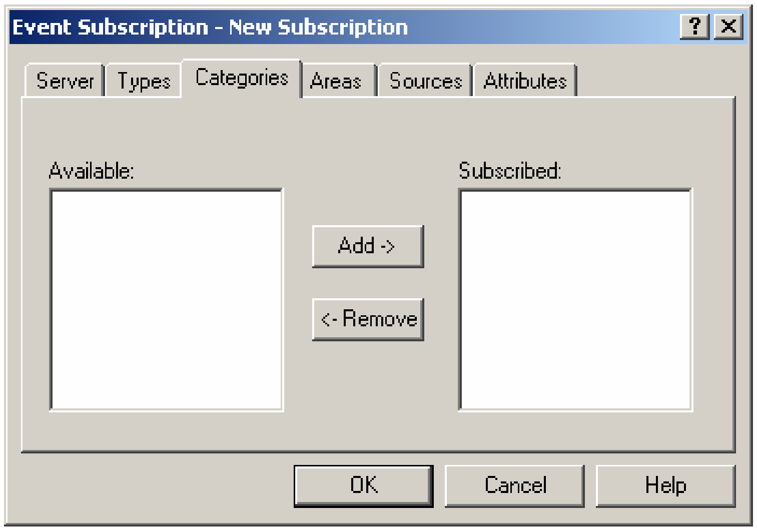


Figure 12. Alarm Subscription: Categories Tab

Areas

To select an area or group of areas for filtering in a subscription, click **Browse** in the **Areas** tab, shown below. The OPC Event Server Area/ Source Browser will appear listing all available Areas for your Event Server. Area subscription supports wildcards. Wildcards subscribe to the format of Visual Basic's "like" command. For example, Area1* will subscribe to all alarm areas that contain strings beginning with Area1. Area1* will subscribe to the root area and its child areas. A detailed explanation of the wildcard support can be found in the OPC Alarm and Events documentation. It is a good idea to thoroughly read the wildcard documentation before attempting to use complicated expressions. It is also possible to delete an area from this same window.

To delete an area, select it from the list of areas for this particular subscription and click the **Delete** button. Not all OPC AE servers support area filtering as part of the subscription.

Note
If no areas are listed, then all areas are selected by default.

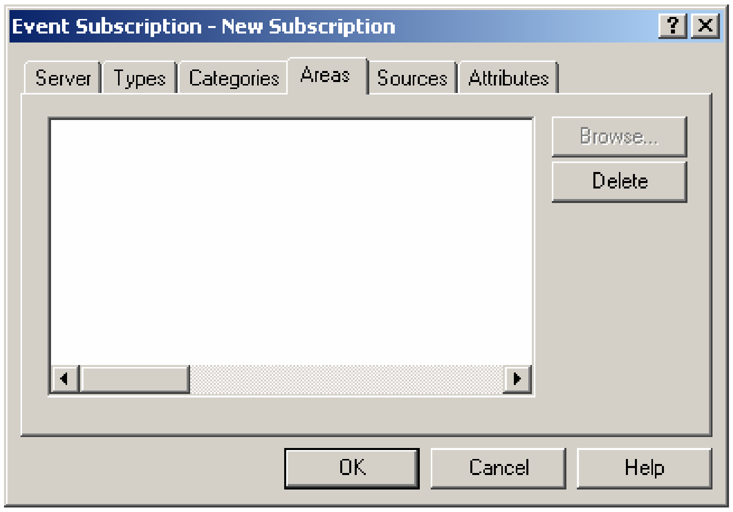


Figure 13. Alarm Subscription: Areas Tab

Sources

The **Sources** tab, shown below, allows you to filter on a source or group of sources for a particular subscription. To select a source, click **Browse** and select one from the OPC Event Server Area/ Source Browser. The source subscription provides the wildcard support found in the area subscription.

It is also possible to delete a source for a particular subscription. To delete a source, select it from the list of sources for this particular subscription and click **Delete.** Not all OPC AE servers support source filtering as part of the subscription.

Note
If no areas are listed, then all areas are selected by default.

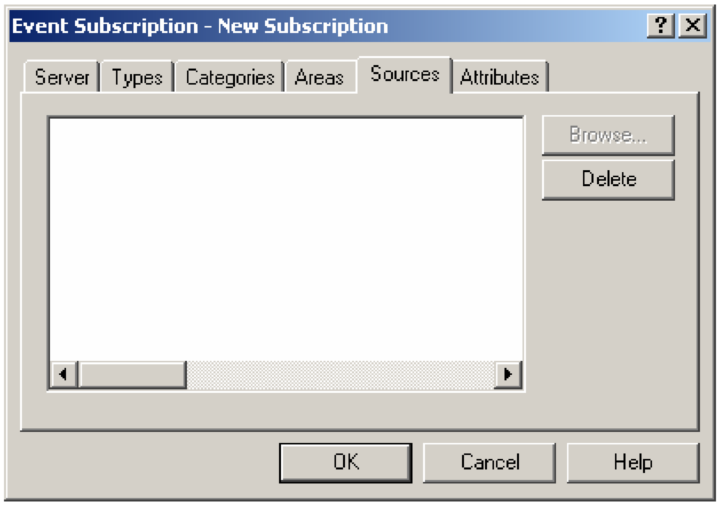


Figure 14. Alarm Subscription: Sources Tab

Attributes

The **Attributes** tab, shown below, allows you to add extra attributes to a particular event category within a subscription. Select the event category from the corresponding drop-down menu. Choose the desired items from the available attributes list, and then click **Add** button.

To remove an attribute from the subscribed list, select that particular attribute and then click **Remove.** It is important to note that the order of the subscribed attributes does matter, because the first attribute listed will have priority over all other attributes in the list. The order of the attributes also determines the order in which they will be selected from the server. To change the order of the attributes in the subscribed field, simply select an attribute and click the "up" and "down" arrow buttons. Server-specific information will be displayed in the Attributes1-n columns. The attributes columns are added in the column property page.

Note
To receive extra attributes you must request them per event category.

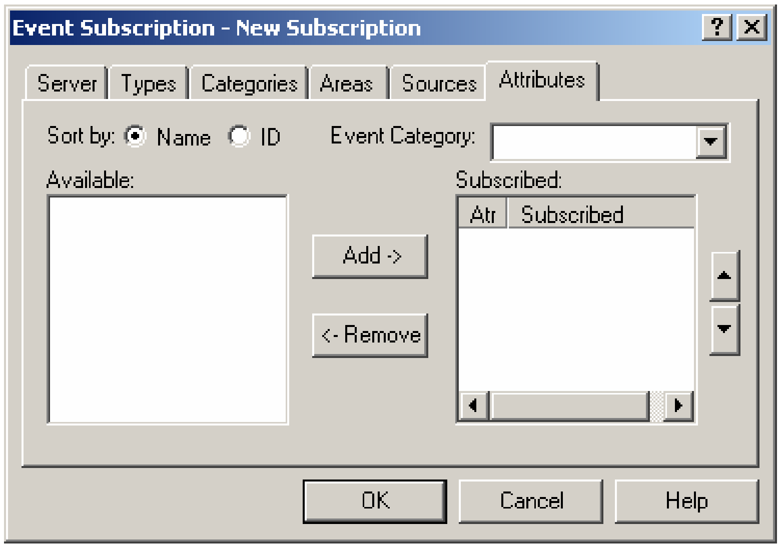


Figure 15. Alarm Subscription: Attributes Tab

Different Types of Scripts

There are four basic types of scripts that can be inserted into the user interface: global, periodic, event, and alarm. Which type you choose will depend on how you want the scripts to be triggered.

General

Inserting a Script

To add a new script, select **Insert New Trigger** from the **Edit** menu. The type of script inserted depends on which type is selected in the tree view of the user interface. You can also right-click the script type in the tree view and select **Insert Trigger,** as shown below. Alternatively, you can also click the **Insert Script** button on the **Data-Manipulation** toolbar.

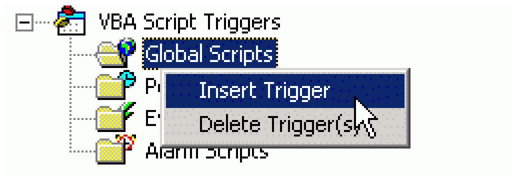


Figure 16. Inserting a Trigger

If no type is selected, the **Insert Script** dialog box will open as shown below, allowing you to choose a script type.

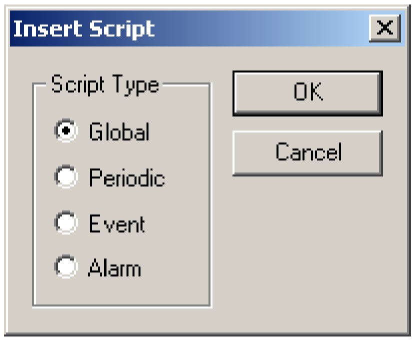
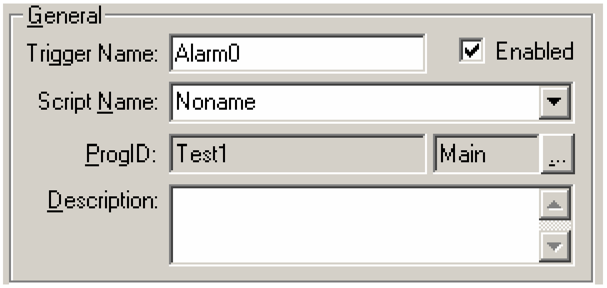


Figure 17. Insert Script Dialog Box

Regardless of the type of script, the following information will appear on the top of the script configuration window in the right-hand pane of the ScriptWorX screen.



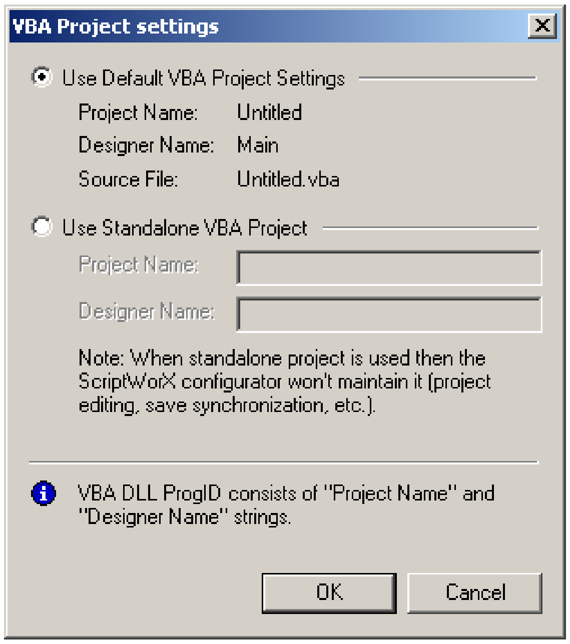
Trigger Name: Alarm0
[✓] Enabled
Script Name: Noname
ProgID: Test1 Main

Figure 18. General Information for All Scripts

The **Trigger Name** is automatically created, and it depends on the type of script that is being created. The created names consist of the type of script (global, periodic, event, or alarm) followed by an incremental number starting with 0. You can change the **Script Name** if you like by simply type the new name in the appropriate field and then clicking the **Apply.**

Note
If you change the configuration of a script, you must click **Apply** to have these changes actually applied. If you try to change to another configuration after making changes, you will be prompted to save the changes.

You can also enter a description of what the script will actually do when it is running. The **ProgID** field allows you to enter the project and designer names with which this script is associated. Clicking the button to the right of this field opens the **VBA Project Settings** dialog box, shown below. You can use the default project settings, or you can specify the project and designer names for the script by using a standalone VBA project.



● Use Default VBA Project Settings
Project Name: Untitled
Designer Name: Main
Source File: Untitled.vba

○ Use Standalone VBA Project
Project Name:
Designer Name:

Note: When standalone project is used then the ScriptWorX configurator won't maintain it (project editing, save synchronization, etc.).

VBA DLL ProgID consists of "Project Name" and "Designer Name" strings.

Figure 19. VBA Project Settings Dialog Box

Date/Time Range

The **Date/Time Range** section of the trigger properties, shown below, allows you to set when you want the script to start and stop running.

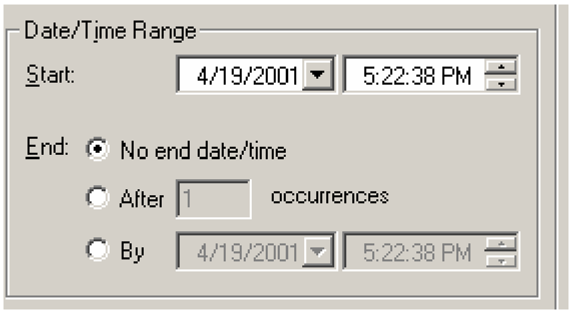

Date/Time Range
Start: 4/19/2001 5:22:38 PM
End: (●) No end date/time
( ) After 1 occurrences
( ) By 4/19/2001 5:22:38 PM

Figure 20. Date/Time Range for Scripts

Other General Features

The last set of general features common to all types of scripts are the four buttons, shown below, that can be found at the bottom of the trigger section for each script.

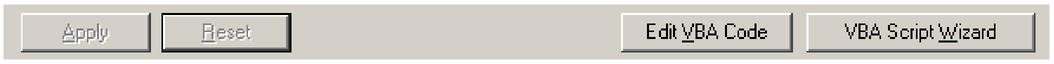


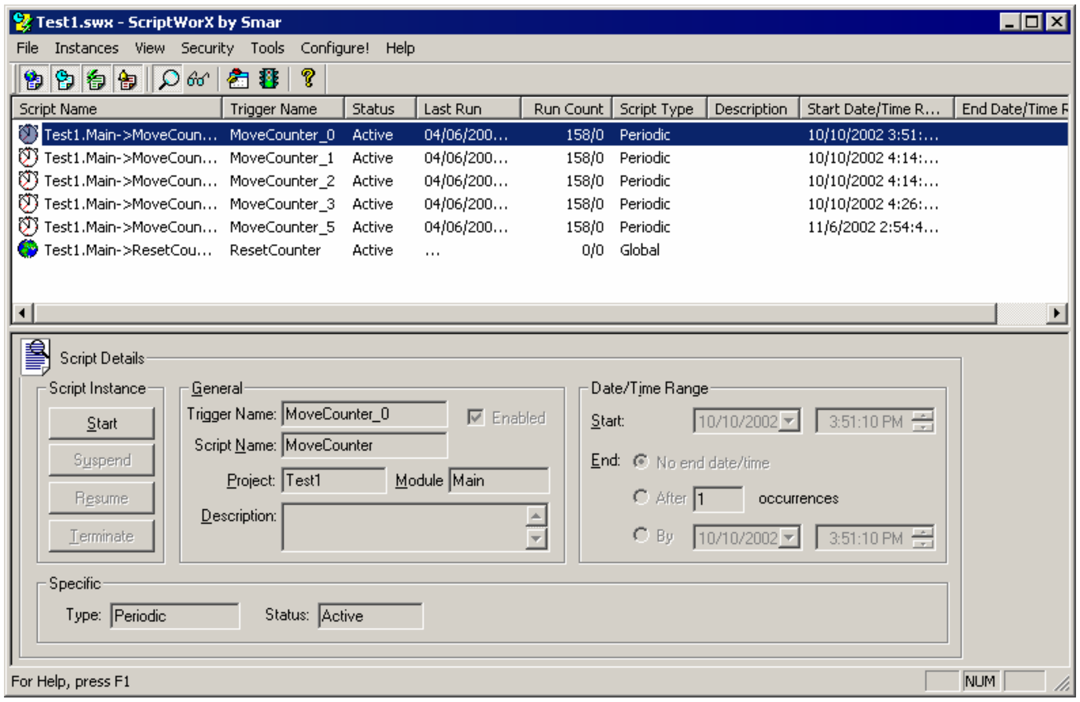
Figure 21. User Interface Action Buttons

The **Apply** and **Reset** buttons deal with changes that you have made to the configuration of the scripts. To save the changes you have made to the script, click **Apply.** Clicking the **Reset** button will restore the script to the last saved configuration.

The other buttons refer to Visual Basic for Applications, and provide simple methods of accessing the actions and applications needed for writing the code behind each script. The **Edit VBA Code** button launches the VBA Editor, which allows you to enter new or additional code for the current script. The **VBA Script Wizard** launches the Smar Visual Basic for Applications Script Wizard, which contains many script templates for the most common operations. This feature will be discussed later in greater detail.

Common Runtime Features

To enter runtime mode, save the current configuration to a file and then select **Runtime** from the menu bar. The following screen will be displayed. All scripts are displayed in the top pane of the screen.


Test1.swx - ScriptWorX by Smar
File Instances View Security Tools Configure! Help

Script Name	Trigger Name	Status	Last Run	Run Count	Script Type	Description	Start Date/Time R...	End Date/Time F
Test1.Main->MoveCoun...	MoveCounter_0	Active	04/06/200...	158/0	Periodic		10/10/2002 3:51:...	
Test1.Main->MoveCoun...	MoveCounter_1	Active	04/06/200...	158/0	Periodic		10/10/2002 4:14:...	
Test1.Main->MoveCoun...	MoveCounter_2	Active	04/06/200...	158/0	Periodic		10/10/2002 4:14:...	
Test1.Main->MoveCoun...	MoveCounter_3	Active	04/06/200...	158/0	Periodic		10/10/2002 4:26:...	
Test1.Main->MoveCoun...	MoveCounter_5	Active	04/06/200...	158/0	Periodic		11/6/2002 2:54:4...	
Test1.Main->ResetCou...	ResetCounter	Active	...	0/0	Global			

Script Details

Script Instance: Start, Suspend, Resume, Terminate

General
Trigger Name: MoveCounter_0 [✓ Enabled]
Script Name: MoveCounter
Project: Test1 Module Main
Description:

Date/Time Range
Start: 10/10/2002 3:51:10 PM
End: [✓ No end date/time] After 1 occurrences By 10/10/2002 3:51:10 PM

Specific
Type: Periodic Status: Active

For Help, press F1 NUM

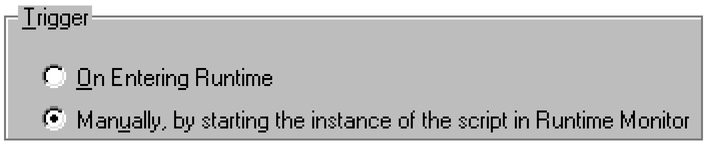
Figure 22. Runtime Mode Screen

When you select a script, the script details are displayed in the bottom pane of the screen. All fields in the script details will automatically be filled in according to the configuration elements settings. If you would like to start another instance of the script, simply click the **Start Instance** button. This button is the same button you would use to trigger the script manually.

The list box shown above will post all scripts that have been configured as well as their status. Using the command buttons on the left of the **Script Details** section, you can **Start, Suspend, Resume,** or **Terminate** an instance of a script. These buttons are helpful if a script is running improperly, or if you have a script running on a certain interval and you need to change it or run it again.

Global Scripts

The use of **global scripts** indicates that you want to run the script either at the first instance of entering runtime or by triggering it manually. Once the global script is inserted into the user interface, the configuration screen will appear in the right-hand pane. The **General** and **Date/Time Range** sections have already been discussed in the previous section. The other configuration section is **Trigger,** shown below.


Trigger

On Entering Runtime [✓ Manually, by starting the instance of the script in Runtime Monitor]

Figure 23. Trigger Configuration for Global Script

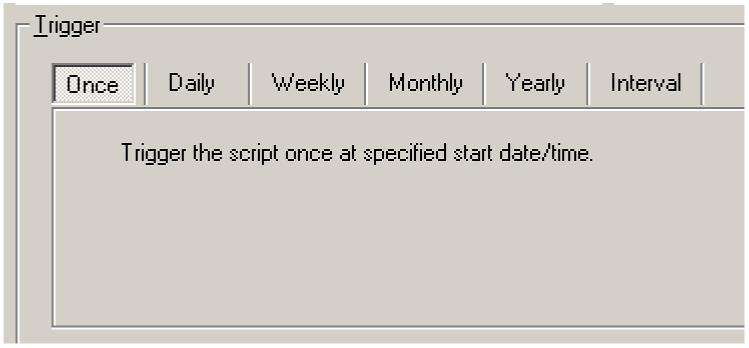
The **Trigger** section determines how you want global scripts to be triggered. If the **On Entering Runtime** option is selected, the script will start when the display enters runtime mode. If you do not want this to happen, you can select the **Manually...** option, which allows you to run the script manually in runtime mode by clicking the **Start Instance** button in the Runtime Monitor.

Periodic Scripts

Periodic scripts are set to trigger after some time restriction or task reoccurrence. There are a few more ways to configure periodic scripts than with global scripting. You should first select how you want to trigger the script: once, daily, monthly, yearly, or at some interval. After that, you should set the corresponding trigger conditions.

Once

When **Once** is selected for a script, as shown below, the script is launched once at the specified start time and runs until it is completed. To get this script triggered again, it must be done manually. This is convenient if you want to print out a long report but want to do it after everyone has left so as to not clog the printer. Simply write a script and set it to start at a time when you know it will not be inconveniencing anyone. It is also helpful if you want to test a script to make sure that it is working properly.

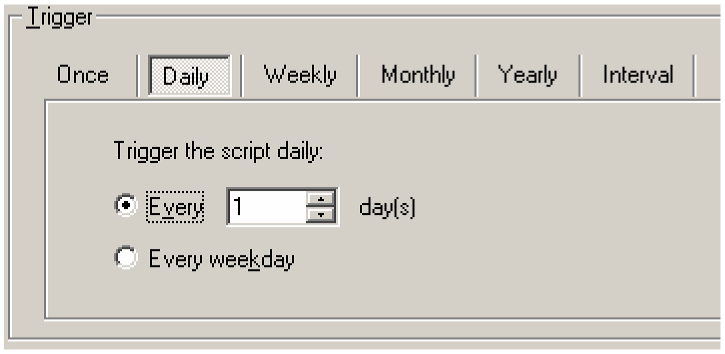


Trigger the script once at specified start date/time.

Figure 24. Periodic Scripting Configuration - Once

Daily

As shown below, setting a script to **Daily** gives you two options: **Every *x* day(s)** or **Every weekday.** If you select the first option, then you must enter the number of days between each trigger. For example, if you wanted to trigger an event to occur every other day, you would enter '2' for the number of days. If you select **Every weekday,** the script will run every Monday, Tuesday, Wednesday, Thursday, and Friday.



Trigger the script daily:

(●) Every 1 day(s)

( ) Every weekday

Figure 25. Periodic Scripting Configuration - Daily

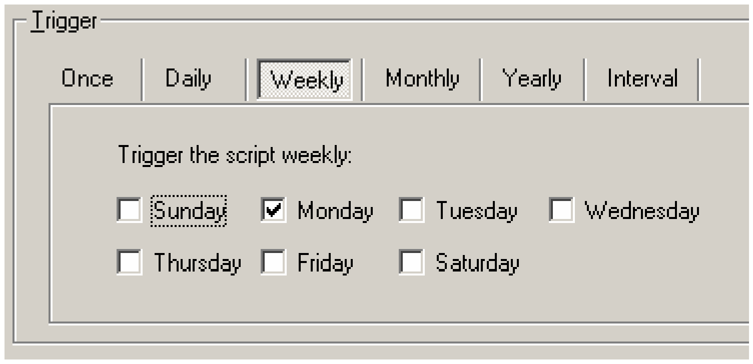
In addition to these two options, the trigger for the script can be further configured by selecting not only a start time but an end option as well in the **Date/Time Range** section. There are three different end options, which are listed in the table below.

End Option	Result
No end date / time	Once triggered, the script will not end.
End after *n* occurrences	After the triggered script has run for *n* times, it will stop.
End by (selected date and time)	The triggered script will stop at the selected date and time.

You are also required to enter a start date/time to indicate when you want the script to start.

Weekly

The **Weekly** trigger options, shown below, are relatively straightforward. Simply select the day(s) on which you want the script to run. It is possible to select more than one day. As stated in the previous section, you can further configure the trigger for the script by selecting a start date/time and one of the three available end options in the **Date/Time Range** section.



Trigger the script weekly:

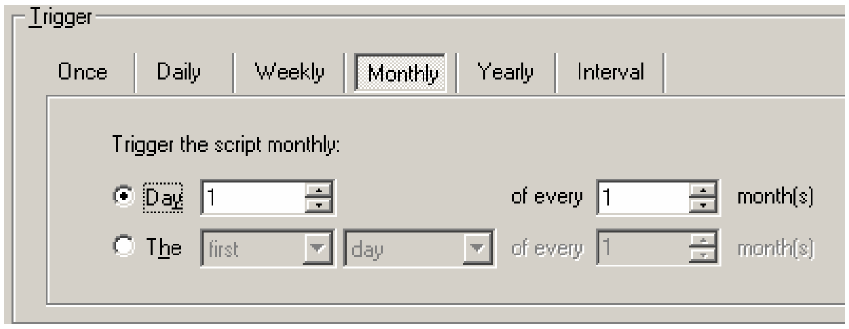
☐ Sunday ☑ Monday ☐ Tuesday ☐ Wednesday
☐ Thursday ☐ Friday ☐ Saturday

Figure 26. Periodic Scripting Configuration - Weekly

Monthly

There are two options to consider when setting a **Monthly** trigger for a script, as shown below. The first option allows you to select the day of the month on which you want to set the trigger. The second option allows you to determine how frequently the script will be triggered (i.e. every month, every two months, etc.). If the second option is chosen, you must enter the [first, second, third, fourth or last], [day, weekday, weekend day, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday or Saturday] of every [*x*] month(s).

Using the monthly trigger can be useful for scheduling events that need to occur every month or every quarter. These monthly triggers can also be affected by the start and end settings defined in the **Date/Time Range** section.



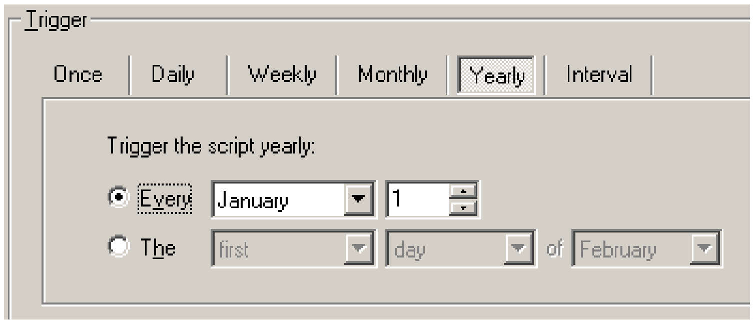
Trigger the script monthly:

● Day 1 of every 1 month(s)
○ The first day of every 1 month(s)

Figure 27. Periodic Scripting Configuration - Monthly

Yearly

When setting a script to trigger **Yearly,** as shown below, you have two options: The first option is every [month] on the [day]. The second option is the [first, second, third, fourth, or last], [day, weekday, weekend day, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, or Saturday] of [month]. This is very helpful if you need to run a year-end report, or you need to perform a task once a year and you want to schedule it in advance.



Trigger the script yearly:

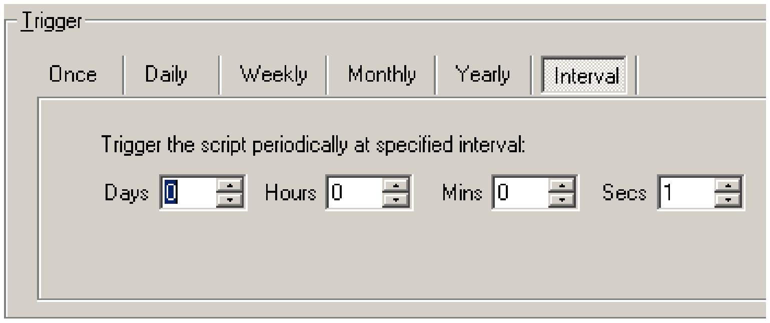
[✓ Every] January 1

The first day of February

Figure 28. Periodic Scripting Configuration - Yearly

Interval

The final trigger option for periodic scripts is the **Interval** option, shown below. When this option is selected, the script can be triggered to run after a set period of time. This is a useful option if you want to update data once in a while. Simply set the interval in the trigger area [Day(s)] [Hour(s)] [Minute(s)], and [Second(s)].



Trigger the script periodically at specified interval:

Days 0 Hours 0 Mins 0 Secs 1

Figure 29. Periodic Scripting Configuration - Interval

Event Scripts

Event scripts are scripts that are triggered after an event has occurred. The configuration screen, shown below, contains the following three options:

- First time condition becomes TRUE
- Always when expression becomes TRUE
- Continuously when expression is TRUE

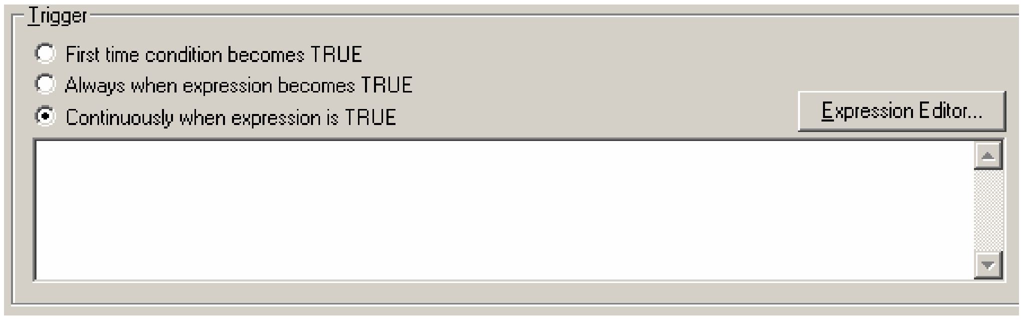

Trigger
First time condition becomes TRUE
Always when expression becomes TRUE
Continuously when expression is TRUE
Expression Editor...

Figure 30. Events Script Configuration Screen

The condition/expression to which each of these options refers is set either by using the **Expression Editor** or by manually entering text in the blank expression box. Clicking the **Expression Editor** button opens the **Edit Expression** dialog box, shown below.

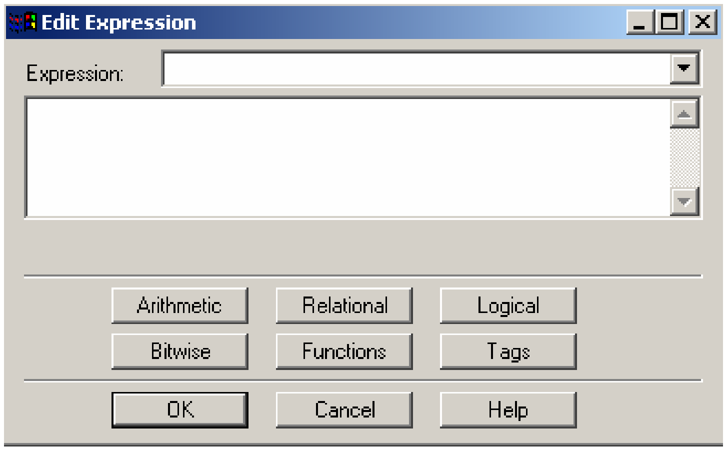


Figure 31. Edit Expression Dialog Box

Point Extension Syntax

The **Point Extension Syntax (PES)** allows for retrieving additional information related to OPC tags, such as quality and timestamp. The following are example expressions using a valid PES request:

- tag:Smar.Simulator\SimulatePLC.Ramp#timestamp
- tag:Smar.Simulator\SimulatePLC.Ramp#quality
- tag:\\pc1\Smar.Simulator\SimulatePLC.Ramp#timestamp
- tag:\\pc1\Smar.Simulator\SimulatePLC.Ramp#quality

Sometimes it may be necessary to enforce the “request data type” to a specific type, such as “string,” in order to display this information in a process point.

This dialog box contains the following options:

- Arithmetic
- Relational
- Logical
- Bitwise
- Functions
- Tags

Arithmetic

Clicking **Arithmetic** in the **Edit Expression** dialog box allows you to perform the following mathematical functions:

The symbols '+', '-', '*', '/' and '%' use the following format:

expression :: parameter **symbol** parameter

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	+ or - or * or / or %

Result

The expression results in a number of any type (float, long, etc.).

Examples

Symbol	Description	Example	Result
`+`	Addition	`~~var1~~ + ~~var2~~`	`8+3 = 11`
`-`	Subtraction	`~~var1~~ - ~~var2~~`	`8-3 = 5`
`*`	Multiplication	`~~var1~~ * ~~var2~~`	`8*3 = 24`
`/`	Division	`~~var1~~ / ~~var2~~`	`8/3 = 2.66667`
`%`	Calculates the remainder after division	`~~var1~~ % ~~var2~~`	`8%3 = 2`
(and)	Gives precedence to parts of the calculation	`~~var1~~ / (~~var2~~ + ~~var3~~)`	`8/(3+2) = 1.6`

Relational

Clicking **Relational** in the **Edit Expression** dialog box allows you to edit expressions using the following:

The symbols '<', '>', '<=', '>=', '==' and '!=' use the following format:

expression :: parameter **symbol** parameter

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	< or > or <= or >= or == or !=

Result

The expression results in a Boolean value (0 or 1).

Examples

Symbol	Description	Example	Result
`<`	Less than	`~~var1~~ < ~~var2~~`	`8<3 = 0`

Symbol	Description	Example	Result
>	Greater than	`~~var1~~ > ~~var2~~`	8>3 = 1
<=	Less than or equal to	`~~var1~~ <= ~~var2~~`	8<=3 = 0
>=	Greater than or equal to	`~~var1~~ >= ~~var2~~`	8>=3 = 1
==	Equal to	`~~var1~~ == ~~var2~~`	8==3 = 0
!=	Not equal to	`~~var1~~ != ~~var2~~`	8!=3 = 1

Logical

Clicking **Logical** in the **Edit Expression** dialog box allows you to edit expressions using the following:

The symbols '&&' and '||' use the following format:

expression :: parameter **symbol** parameter

The symbol '!' uses the following format:

expression :: **symbol** parameter

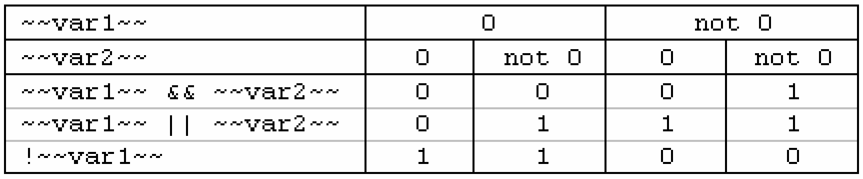
Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	&& or || or !

Result

The expression results in a Boolean value (0 or 1).

Truth table



`~~var1~~`	0		not 0	
`~~var2~~`	0	not 0	0	not 0
`~~var1~~ && ~~var2~~`	0	0	0	1
`~~var1~~ || ~~var2~~`	0	1	1	1
`!~~var1~~`	1	1	0	0

Examples

Symbol	Description	Example	Result
&&	And	`~~var1~~ && ~~var2~~`	8 && 3 = 1
||	Or	`~~var1~~ || ~~var2~~`	8 || 3 = 1
!	Not	`!~~var1~~`	!8 = 0

Bitwise

Clicking **Bitwise** in the **Edit Expression** dialog box allows you to edit expressions using the following:

The symbols '&', '|', and '^' of the bitwise group use the following format:

expression :: parameter **symbol** parameter

The symbol '~' of the logical group uses the following format:

expression :: **symbol** parameter

The symbols 'shl' and 'shr' of the bitwise group use the following format:

expression :: **symbol** (value, shift by)

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	&& or || or ^ or shl or shr or ~

Result

The expression results in a number when the parameters used contain numbers.

Bit Table

	Binary (Decimal)	Binary (Decimal)
`~~var1~~`	`0000.0000.0000.1000(8)`	`0000.0000.0110.0000(96)`
`~~var2~~`	`0000.0000.0000.1010(10)`	`0000.0000.0000.1000(8)`
`~~var1~~ & ~~var2~~`	`0000.0000.0000.1000(8)`	`0000.0000.0000.0000(0)`
`~~var1~~ | ~~var2~~`	`0000.0000.0000.1010(10)`	`0000.0000.0110.1000(104)`
`~~var1~~ ^ ~~var2~~`	`0000.0000.0000.0010(2)`	`0000.0000.0110.1000(104)`
`shl (~~var1~~,3)`	`0000.0000.0100.0000(64)`	`0000.0011.0000.0000(768)`
`shr (~~var1~~,3)`	`0000.0000.0000.0001(1)`	`0000.0000.0000.1100(12)`
`~(~~var1~~)`	`1111.1111.1111.0111(-9)`	`1111.1100.1111.1111(-97)`
`bittest(~~var1~~,3)`	`0000.0000.0000.0001(1)`	`0000.0000.0000.0000(0)`

Examples

Symbol	Description	Example	Result
`&`	Bit And	`~~var1~~ & ~~var2~~`	`8 && 3 = 0`
`|`	Bit Or	`~~var1~~ | ~~var2~~`	`8 || 3 = 11`
`^`	Bit eXclusive Or	`~~var1~~ ^ ~~var2~~`	`8^3=11`
`shl`	Bit shift left	`shl(~~var1~~,3)`	`8<<3=64`
`shr`	Bit shift right	`shr(~~var1~~,3)`	`8>>3=1`
`~`	Not (two's complement)	`~(~~var1~~)`	`!8 = -9`
`bittest`	Bit Test	`bittest ( 5 , 0 )`	`1`

Note
The bittest function requires you to specify the position of the bit to be tested. You must indicate that it starts from 0. In other words, a bit position of "0" indicates the "less significant" bit.

Functions

Clicking **Functions** in the **Edit Expression** dialog box allows you to edit expressions using the following:

The symbols 'sin', 'asin', 'cos', 'acos', 'tan', 'atan', 'log', 'ln', 'exp', 'sqrt', 'abs', 'ceil', and 'floor' use the following format:

expression :: **symbol** (parameter)

The symbols 'pow', 'min', and 'max' use the following format:

expression :: **symbol** (parameter,parameter)

The symbol 'if' uses the following format:

expression :: **symbol** (parameter,parameter,parameter)

Where

Parameter	A local variable, an OPC tag, a constant, or another expression
Symbol	sin, asin, cos, acos, tan, atan, log, ln, exp, sqrt, abs, ceil, floor, min, max, pow, or if

Result

The expression results in a number.

Examples

Symbol	Description	Example	Result
`sin`	sine of an angle in radians	`sin(~~var1~~)`	`sin(0.785)=0.71`
`cos`	cosine of an angle in radians	`cos(~~var1~~)`	`cos(0.785)=0.71`
`tan`	tangent of an angle in radians	`tan(~~var1~~)`	`tan(0.785)=1.0`
`asin`	arc sine returns an angle in radians	`asin(~~var1~~)`	`asin(0.5)=0.52`
`acos`	arc cosine returns an angle in radians	`acos(~~var1~~)`	`acos(0.5)=1.05`
`atan`	arc tangent returns an angle in radians	`atan(~~var1~~)`	`atan(1)=0.785`
`sqrt`	Returns the square root	`sqrt(~~var1~~)`	`sqrt(100)=10`
`pow`	Returns value 1 raised to the power value 2	`pow(~~var1~~,~~var2~~)`	`pow(100,1.5)=1000`
`log`	10 based	`log(~~var1~~)`	`log(100)=2`

Symbol	Description	Example	Result
	logarithm		
`ln`	*e* based logarithm	`ln(~~var1~~)`	`ln(7.389)=2`
`exp`	Exponential	`exp(~~var1~~)`	`exp(2)=7.389`
`abs`	Absolute value	`abs(~~var1~~)`	`abs(-1)=1`
`ceil`	Integer ceiling	`ceil(~~var1~~)`	`ceil(7.39)=8`
`floor`	Integer floor	`floor(~~var1~~)`	`floor(7.39)=7`
`min`	Lowest value of two	`min(~~var1~~,~~var2~~)`	`min(10,5)=5`
`max`	Highest value of two	`max(~~var1~~,~~var2~~)`	`min(10,5)=10`
`if`	Conditional statement	`if(~~var1~~<~~var2~~, ~~var1~~,~~var2~~)`	`if(5<8,5,8)=5`
`like`	Wildcard string compare	`Like(string, pattern, casesensitive')`	
`quality`	Quality of tag or expression	`See below.`	`See below.`
`tostring`	Type conversion	`See below.`	`See below.`
`0x`	Hexadecimal constant	`x=0x11`	17
`0t`	Octal constant	`x=0t11`	9
`0b`	Binary constant	`x=0b11`	3

Note
For the like operator: "string" equals the string to search in; "pattern" equals the string to search for (can include wildcards); nonzero for case-sensitive search; zero for case-insensitive search. String syntax is $"string"$.

You can use these special characters in pattern matches in string:

- ? Any single character.
- Zero or more characters.
- # Any single digit (0-9).
- [charlist] Any single character in charlist.
- [!charlist] Any single character not in charlist.

Quality

The **quality** option on the **Functions** menu of the **Expression Editor** is used to evaluate the quality of an OPC tag or an expression.

The following general syntax is used for quality expressions:

x=quality(expression)

Note
The "(expression)" can also be a simple expression composed of a single tag.

The **quality** function returns the OPC quality of the string between parentheses as one of the following results:

- 192: quality is GOOD
- 64: quality UNCERTAIN
- 0: quality BAD

Note: The OPC Foundation establishes the value ranges for quality. There are actually varying degrees of quality:

- GOOD: 192-252
- UNCERTAIN: 64-191
- BAD: 0-63

For more information, refer to the *OPC Data Access Custom Interface Standard* available for download at the OPC Foundation's Web site, www.opcfoundation.org/.

Example Quality Expression

Expression	Result
x=quality({{Smar.Simulator.1\SimulatePLC.PumpStatus}})	192 (Quality GOOD)

The quality of an expression is determined through the evaluation of each single tag in the expression. Thus, if you have multiple tags in an expression (and each tag has a different quality), the result of the expression (i.e. 192 [GOOD], 64 [BAD], or 0 [UNCERTAIN]) corresponds to the quality of the tag with the lowest quality. If an expression contains a conditional statement (e.g. if, then, or else), then the result of the expression is affected only by the quality of the branch being executed.

Consider the following sample expression:

x= if (quality({{Tag1}}) == 192, {{Tag1}}, {{Tag2}})

This expression can be read as follows:

"If the quality of Tag1 is GOOD (i.e. 192), then the expression result (x) is the value of Tag1. In all other cases (i.e. the quality of Tag1 is UNCERTAIN or BAD), the expression result (x) is the value of Tag2."

We can calculate the results for this expression using different qualities for Tag1 and Tag2, as shown in the figure below.

Case	Tag1 quality	Tag2 quality	Result	Result quality
1	GOOD	GOOD	Tag1	192 (GOOD)
2	GOOD	UNCERTAIN	Tag1	192 (GOOD)
3	GOOD	BAD	Tag1	192 (GOOD)
4	UNCERTAIN	GOOD	Tag2	192 (GOOD)
5	UNCERTAIN	UNCERTAIN	Tag2	64 (UNCERTAIN)
6	UNCERTAIN	BAD	Tag2	0 (BAD)
7	BAD	GOOD	Tag2	192 (GOOD)
8	BAD	UNCERTAIN	Tag2	64 (UNCERTAIN)
9	BAD	BAD	Tag2	0 (BAD)

In cases 1-3 above, the quality of Tag1 is GOOD, and therefore the result of the expression is GOOD. Thus, the result of the expression is not affected by the quality of Tag2 (the “else” branch of the expression), which is ignored.

In cases 4-6, the quality of Tag1 is UNCERTAIN, and therefore the result of the expression is the quality of Tag2.

In cases 7-9, the quality of Tag1 is BAD, and therefore the result of the expression is the quality of Tag2.

Note
The “quality()” function returns a value that represents the quality of the expression within the parentheses but is always GOOD_QUALITY. For example, if Tag1 is BAD_QUALITY then the expression “x=quality({{Tag1}})" will return 0 with GOOD_QUALITY.

The result of an expression is the minimum quality of the evaluated tag in the expression and is affected only by the quality of the conditional (if, then, or else) branch that is executed.

Consider the following sample expression:

x= if ({{TAG_01}}>0,{{TAG_02}},{{TAG_03}})

This expression can be read as follows:

"If the value of TAG_01 is greater than 0, then the expression result (x) is TAG_02. If the value of TAG_01 is less than or equal to 0, then the expression result (x) is TAG_03."

Let's assume that the following values and qualities for these tags:

TAG_01=5 with quality GOOD

TAG_02=6 with quality UNCERTAIN

TAG_03=7 with quality BAD

Because the value of TAG_01 is 5 (greater than 0), the expression result is TAG_02. Thus, the final expression result is 6, and the final expression quality is UNCERTAIN.

Type Conversion

The **tostring** option on the **Functions** menu of the **Expression Editor** takes the value of whatever item is in parentheses and converts it into a string as follows:

The value is +(value)+unit

It can be used to convert from number to string, and it can be very useful for string concatenation.

The proper syntax for the **tostring** option is:

x=$"The value is "$ + tostring(value) + $" unit"$

Note
In the expression above, the word "unit" is placeholder text for a user-specified unit of measurement or variable (e.g. Watt, inches, meters, etc.).

Example Expressions Type Conversion

Expression	Result
x=$"The value is "$ + tostring({{gfwsim.ramp.float}}) + $" Watt"$	“The value is 543.2345152 Watt”

Constants

The **Functions** menu of the **Expression Editor** supports constant values, including hexadecimal, octal, and binary formats.

Example Expressions Using Constants

Expression	Result
x=0x11	17
x=0t11	9
x=0b11	3

The **Expression Editor** conveniently inserts the 0x and 0t and 0b prefixes for you so do not have to recall them.

Interpreting and Translating Constants

The examples below show how values are calculated for each type of constant.

- **Hexadecimal:** 0x20A = 2 * (16^2) + 0 * (16^1) + 10 * (16^0) = 2*256 + 0*16 + 10 * 1 = 512 + 0 + 10 = 522
- **Octal:** 0t36= 3 * (7^1) + 6 *(7^0) = 3* 7 + 6* 1= 21 + 6 = 27
- **Binary:** 0b110 = 1 * (2^2) + 1 * (2^1) + 0 * (2^ 0) = 1 * 4 + 1 * 2 + 0 * 1 = 4+2+0 = 6

Tags

Clicking the **Tags** button in the **Edit Expression** dialog box opens the **OPC Universal Tag Browser,** shown below. You can use the tree view on the left to browse for different OPC servers and to find the tag that you want to include as your condition to trigger the event script.

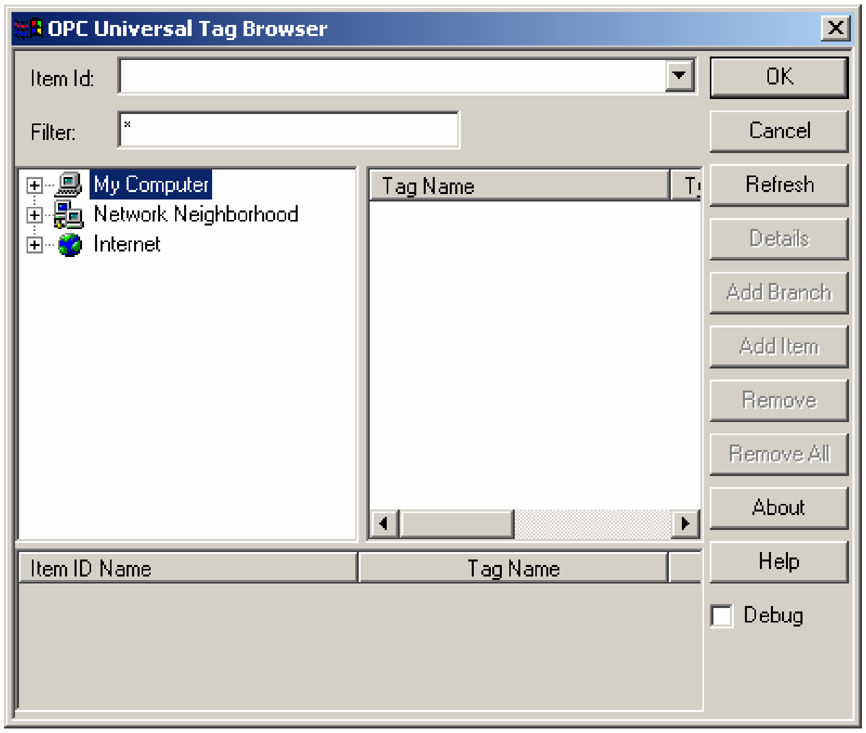
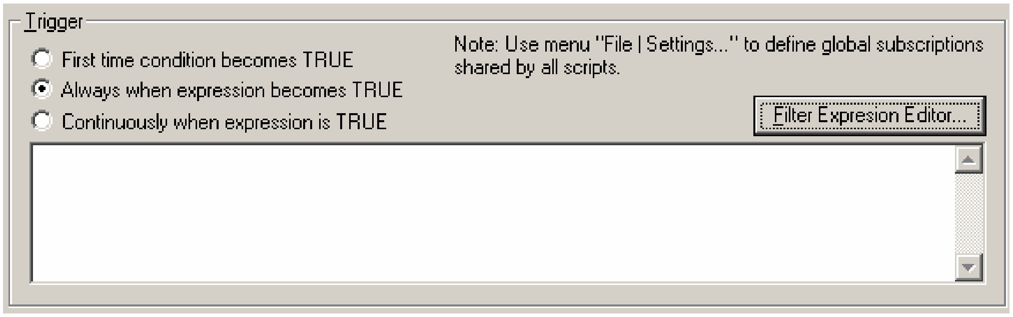


Figure 32. OPC Universal Tag Browser

Alarm Scripts

Alarm scripts are scripts that are triggered after an alarm has occurred. Alarm scripts are set up identically to event scripts. The configuration screen, shown below, contains the following three options:

- First time condition becomes TRUE
- Always when expression becomes TRUE
- Continuously when expression is TRUE



( ) First time condition becomes TRUE
[✓ Always when expression becomes TRUE]
( ) Continuously when expression is TRUE

Note: Use menu "File | Settings..." to define global subscriptions shared by all scripts.

Figure 33. Alarm Script Configuration Screen

The condition/expression to which each of these options refers is set either by using the **Expression Editor** or by manually entering text in the blank expression box. Clicking the **Filter Expression Editor** button opens the **Edit Expression** dialog box, which contains the following options: Arithmetic, Relational, Logical, Bitwise, Functions, and Tags. With the exception of the **Tags** option, all of these options are explained in the **Event Scripts** section above.

Clicking the **Tags** button for alarm scripts gives you the following options, as shown below.

Filter Wizard

Clicking **Filter Wizard** opens the **Filter Wizard** dialog box, shown below, which allows you to choose filters for alarm types and sub conditions. The **Filter Wizard,** shown in the figure below, allows you to choose from the following to items enter in your expression. Select one or more items, and then click **OK.** The filter string is automatically inserted into the **Edit Expression** dialog box.

- **Alarm Types:** Alarm, Ack, Unack, Tracking, and Operator
- **Subconditions:** LoLo, Lo, Hi, HiHi, ROC, and Digital

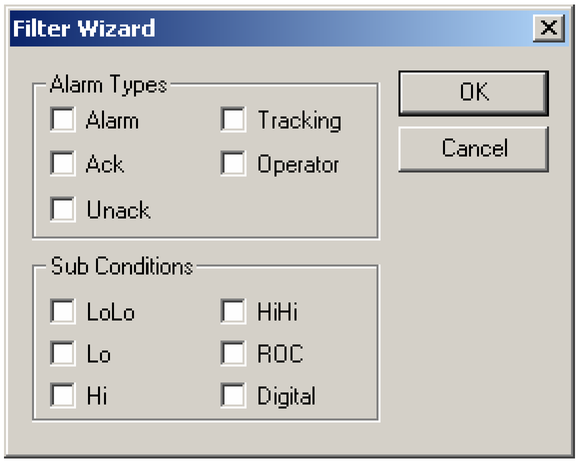


Figure 34. Filter Wizard

Selecting Alarm Attributes

Clicking **Advanced** opens the **Alarm Tag** selection box, which allows you to choose alarm attributes for your alarm filter. Select the attribute that you want to include in the filter expression and click **OK.**

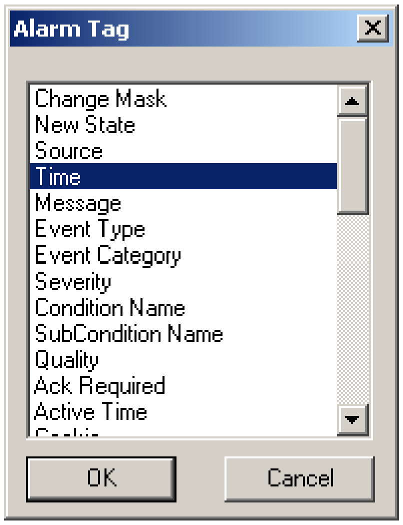


Figure 35. Alarm Tag Selection Box

There are two additional attributes available for use in filtering: **Alarm Type** and **Current Time.** The Alarm Type attribute allows you to filter alarms according to ALARM 1, ACK 2, UNACK 3, OPER 4, TRACK 5 or NORM 6. For example, you can set up a filter with the condition:

X = {{AlarmType}}

If the **Alarm Type** is true, then the alarms are displayed. If they are false then, the alarms are not displayed.

The **Current Time** attribute allows you to filter according to the current time. Only alarms occurring around the current time will be displayed.

Example Alarm Filters

Expression	Result
X = {{Severity}} > 500.	Only alarm messages with a severity greater than 500 will be visible.
X = Like({{Source}}, $"Tag"$,0)	Only messages with the tag in the source name will be displayed.
X = 1.	Filter displays all messages.
X = 0.	Filter does not display any messages.

All filters resolve to TRUE or FALSE. All nonzero values resolve to TRUE.

For more information, please see the AlarmWorX Server documentation.

For global aliases within the expression, use the following syntax:

<#global_alias_name#>

Example:

x=<#RoomTemperature#>

Selecting **Global Alias Browser** opens the Global Alias Browser, as shown in the figure below. Select a global alias from the Global Alias Browser, which includes all global aliases in the global alias database. This eliminates the need to manually type in the alias name. All global aliases that are configured in the Global Alias Engine Configurator are conveniently available to choose from inside the browser. The tree control of the Global Alias Engine Configurator is mimicked in the tree control of the Global Alias Browser. Select a global alias by double-clicking the alias name (e.g. "Floor" in the figure below). The alias name appears at the top of the browser, which automatically adds the <# and #> delimiters to the alias name. Click the **OK** button.

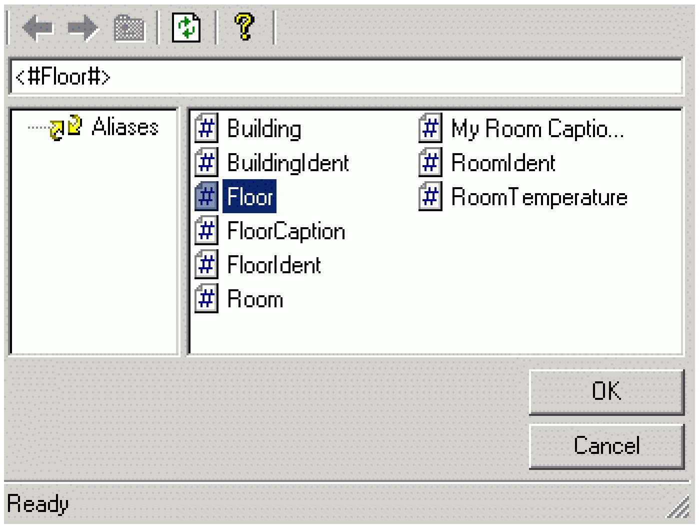


Figure 36. Selecting an Alias From the Global Alias Browser

Automation in ScriptWorX

Script

A **script** is a public subroutine written in the VBA Editor that must be saved as a standalone project. The standalone project is then compiled into a DLL when switching to runtime (automatically). You can also choose to manually compile the DLL at any point by selecting **Make VBA DLL** (Ctrl+B) from the **File** menu in ScriptWorX.

Script Instance

A **script instance** is a running instance of the script. The ScriptWorX32.exe is able to load the VbaMT DLL (VBA multi-threaded DLL) described above, and start a script that is stored in compiled form inside. There can be about 400 instances running concurrently.

ScriptWorX Automation

The ScriptWorX Engine has an automation interface that will be available to any other automation application, such as GraphWorX. The following is a list of methods that are allowed to control and use ScriptWorX.

ScriptWorX 7.0 has a new set of inter-threaded communication objects, which allows you to create your VBA scripts simply and more stable. These objects are very important to the current ScriptWorX design. Every VBA script runs in separated thread, and these VBA scripts needs to communicate somehow.

- **Swx32GlobalStorage** object

It is a set of objects that allows storing and manipulating data between different VBA script threads.

- **IQueue** object

This object implements data queueing capability. One or more VBA scripts are getting data from the queue and the other threads are putting them into. It is a key object to create scripts with thread safe data processing. Typical usage is:

Create one ScriptWorX "infinite" VBA script triggered as global script at the beginning and implement data processing into it (e.g. data logging capability with all the database connections, sending emails, etc.)

Then create one or more VBA scripts driven by periodic events or alarm events, which will put its data into the queue object. These scripts will end up as fast as possible to prevent ScriptWorX threads blocking.

- **ICounter** object

Implements thread-safe counter with upper and lower boundaries.

- **IDataPoint object**

Allows easy OPC data access (it is using GenBroker/GenClient support to access OPC servers).

- **IObjectVariable**

Thread safe automation pointers (IDispatch) distribution.

- **IVariable**

Thread safe update or exchange of VARIANT variables.

- **Swx32Synchronization** object

This object contains a set of two thread synchronization objects.

- **IEvent** object

Thread events with timeouts – one thread is waiting for setting event from other thread.

- **ILock** object

Thread locking with locking timeout.

- Property **ThreadsRunning** as Long

The number of threads currently running.

- Property **LoggerLCID** as Long

The ID number of the logger.

- Property **MessageLCID** as Long

The ID number of the message.

- Function **CreateVariable**(Name as String)

Creates a global variable and initializes it to zero.

- Function **DestroyVariable**(Name as String)

Destroys the specified global variable.

- Function **GetServerTime**(pLowDateTime as Long, pHighDateTime as Long)

Gets the current time.

- Function **GetVariable**(Name as String) as Long

Gets the content of a global variable.

- Function **PrintToConsole**(Message as String)

Prints the message to the ScriptWorX console.

- Function **ResumeScript**(ScriptID as Long)

Resumes script instance ScriptID. ScriptID was returned from StartScript() call in output parameter.

- Function **SetVariable**(Name as String, newVal as Variant)

Sets a global variable to the specified value.

- Function **StartScript**(ScriptName as String, Project as String, Module as String, StrGUID as String, ScriptID as Long) as Long

Starts instance of ScriptName from VBA MT library identified by a DllProgID. The DLL library must be compiled and registered in the VBA Editor. The output parameter ScriptID is used in subsequent calls to identify the script.

- Function **StopThreads**()

Stops all running threads.

- Function **SuspendScript**(ScriptID as Long)

Suspends script instance ScriptID. ScriptID was returned from StartScript() call in output parameter.

- Function **TerminateScript**(ScriptID as Long)

Terminates script instance ScriptID by a soft method. ScriptID was returned from StartScript() call in output parameter. Note this is dangerous operation that can cause loss of data and resource leaks.

- Function **TerminateAllScripts**()

Terminates all running instances of scripts by a soft method.

- Function **TerminateThreads**()

Terminates all running threads by a soft method.

- Function **VariableExists**(Name as String) as Boolean

Returns true if the specified global variable exists.

- Function **Alarm32_SetEvents**(EventList as Variant)

Sets events for Alarm Server.

- Function **Alarm32_GetEvents**(EventListPtr as Variant)

Gets events from Alarm Server.

- Function **Alarm32_SetSubscriptions**(Subscriptions)

Sets subscriptions for Alarm Server.

- Function **Event32_SetEvents**(EventList as Variant) as Long

Sets events for Event32 Server.

- Function **Event32_GetEvents**(EventListPtr as Variant)

Gets events from Event32 Server.

- Function **ExitServer**()

Shuts down the server by closing the ScriptWorX visible client.

- Function **GetSecurity**(SecurityID as Long) as Boolean

Gets security on the specified item.

- Function **Periodic32_SetEvents**(EventList as Variant)

Sets events for Periodic32 Server.

- Function **Periodic32_GetEvents**(EventListPtr as Variant)

Gets events from Periodic32 Server.

- Function **SetConsoleLoggerOptions**(ConsoleLoggerOptions as Long)

Sets the ScriptWorX console logger options.

- Function **SetEventLoggerOptions**(EventLoggerOptions as Long)

Sets the NT event logger options.

- Function **SetMonitorPtr**(pDispatch as Object)

Set back-pointer where the SwxEngine notified events to.

- Function **SetWorkingDirectory**(WorkingDirectory as String)

Sets the ScriptWorX Engine current working directory.

- Sub **StartRuntime**()

Starts runtime mode in ScriptWorX.

- Sub **StopRuntime**()

Returns ScriptWorX to configuration mode.

All of these actions are called from the ScriptWorX user interface but may be used by any other application as well. This means any other application will be able to run and control scripts, obtain monitor notifications about running instances, programs the in-proc servers, and so on. Other applications should only need to use the StartScript() method.

ScriptWorX is registered at GenRegistrar and is thus available to all other applications.

Writing a Script

Introduction

Once you have configured the triggering of the script, it is necessary to write the corresponding script itself. A script is a Visual Basic for Applications (VBA) public subroutine stored in the referenced project and module, as configured in the trigger options in the ScriptWorX user interface.

Note
ScriptWorX supports ScriptWorx/VBA document synchronization and contains has VBA Script Wizards, which greatly simplify the creation of new scripts.

Note
It is strongly suggested that the configuration of a script in the ScriptWorX user interface and the writing of the script be conducted simultaneously. This will help to ensure that the Script, Project, and Module names match up correctly, and will also make testing the script and its configuration much easier.

1. Before writing the script, it is necessary to set the working directory, which is the directory where the compiled .dll will be located and where all configuration files will be saved. To set the working directory, select **Set Working Directory** from the **Tools** menu.

2. When you are ready to write a script, either click on the **Visual Basic Editor** button on the toolbar, or select **Macros - Visual Basic Editor** from the **Tools** menu. This will launch the Visual Basic Editor, where you will write the script.

3. Once in the Visual Basic Editor, make sure the project name and module correspond to the entries made in the ScriptWorX user interface. The script name will correspond to the name used in the actual code, as shown below.

```
Public Sub ScriptName()
        Msgbox "hello"
End Sub
```

Note
All scripts are written in Visual Basic and obey all rules and methods therein. For questions regarding the actual writing of the code, please refer to the Visual Basic documentation.

4. Once the code has been written, save the project by selecting **Save As** from the **File** menu, and enter the same project name as was referenced in the configuration of the script.

5. Compile the VBA file by selecting **Make ProjectName.DLL** from the **File** menu.

6. After the file has been compiled, close the Visual Basic Editor and return to ScriptWorX.

7. Enter runtime mode and watch the scripts execute.

Examples

Several examples of ScriptWorX configurations are provided in the installation under the Smar\ProcessView\Examples directory in the "ScriptWorX Examples" folder. Except for the "Outlook Mail" and "Running Display" examples, each example can be configured using the text file that is associated with the example.

Carousel

The Carousel example opens a GraphWorX display file every time the script is run. The displays to be shown are listed in the text file "Carousel.txt," which is located in the ScriptWorX Examples Directory. Notice that the files listed do not have a path. The example reads a line from the file and pastes the path of GEN32DEMO in front of the line. This means that all displays that you want to add to the Carousel must be located in the GEN32DEMO directory.

To use this example, you must have GraphWorX installed on your computer, as well as the GEN32DEMO Example. Make sure that the GWX Object is selected in the **Tools - References** menu in the VBA Editor.

Customer DB

The Customer DB example cannot be configured. Every time the **launchCustomerForm** method is called, a VBA form appears. Fill in this form and click the **Close** button. The data entered in the form will then be saved in a Microsoft Access Database "CustomerDb.mdb" file using the Microsoft DAO 3.51 Object.

Excel Logger

The Excel Logger example writes OPC tag values to Microsoft Excel using the Excel OLE automation interface. The Excel file and the OPC tags to be logged are stored in the text file "ExcelLogger.txt." The first line in the file is the location of the Excel File where the log should be written. The remaining lines are valid OPC tags.

Note
The module used to retrieve the OPC tag values only supports local OPC servers. To use this Example, you must have Microsoft Excel 97 or 2000 installed on your computer. Make sure that the Microsoft Excel Object (version 8.0 or 9.0) is selected in the **Tools - References** menu in the VBA Editor.

Outlook Mail

The Outlook Mail example writes OPC tag values to a Microsoft Outlook mail message. You can select the tags to log in the configuration form what appears when calling the **showConfiguration** method. In this form, you also can select the recipients and the subject of the message.

Every time the method **sendMail** is called, the example will open Microsoft Outlook, retrieve the current values of the selected OPC tags, and write these values in the e-mail message. When these tasks are completed, the message is sent and Microsoft Outlook is closed.

To use this example, you must have Microsoft Outlook installed on your computer, and the Outlook Object must be selected in the **Tools - References** menu in the VBA Editor.

Word Logger

The Word Logger example writes OPC tag values to Microsoft Word using the Word OLE automation interface. The Word document and the OPC tags to be logged are stored in the text file "WordLogger.txt." The first line in the file is the location of the Word Document where the log should be written. The remaining lines are valid OPC tags.

Note
The module used to retrieve the OPC tag values only supports local OPC servers. To use this example, you must have Microsoft Word 97 or 2000 installed on your computer. Make sure that the Microsoft Word Object (version 8.0 or 9.0) is selected in the **Tools - References** menu in the VBA Editor.

Script Configuration Examples

The following are detailed examples of how to configure scripts using ScriptWorX.

Beep

The first example is very simple and is designed as a test to make sure that ScriptWorX is communicating properly between all of its parts. With the help of Script Wizard and automatic document synchronization, the creation of new script trigger and script itself is an easy task.

Example 1

1. Open a new ScriptWorX file.

2. Insert a new periodic script by right-clicking **Periodic Scripts** on the tree view in the left-hand pane and selecting **Insert Trigger.** Alternatively, select **Insert New Trigger** from the **Edit** menu, or simply click the **Insert Trigger** button on the toolbar.

3. Save the configuration file under desired the name, e.g. Sample.swx.

4. Check that the Project and Module/Designer fields are automatically filled in for you. The Project corresponds to the VBA project, and the Module corresponds to the VBA designer module.

5. Select the **Interval** tab, and make sure that the 1 second interval is already entered.

6. In the configuration screen, change the name to **doBeep** in the **Trigger Name** and **Script Name** fields. The script name corresponds to the name of the script placed in VBA Editor in the related Project and Module.

7. Click **Apply.**

8. Click the **Edit VBA Code** button. The VBA Editor opens, and the script skeleton is already created there:

```
Public Sub DoBeep()
    ' TODO: Add your procedure code here
End Sub
```

9. Enter the body of the script:

Beep

10. Compile the project into .dll by selecting **Make Sample.dll** from the **File** menu in the **VBA Editor.**

11. Click the **Runtime** menu in ScriptWorX.

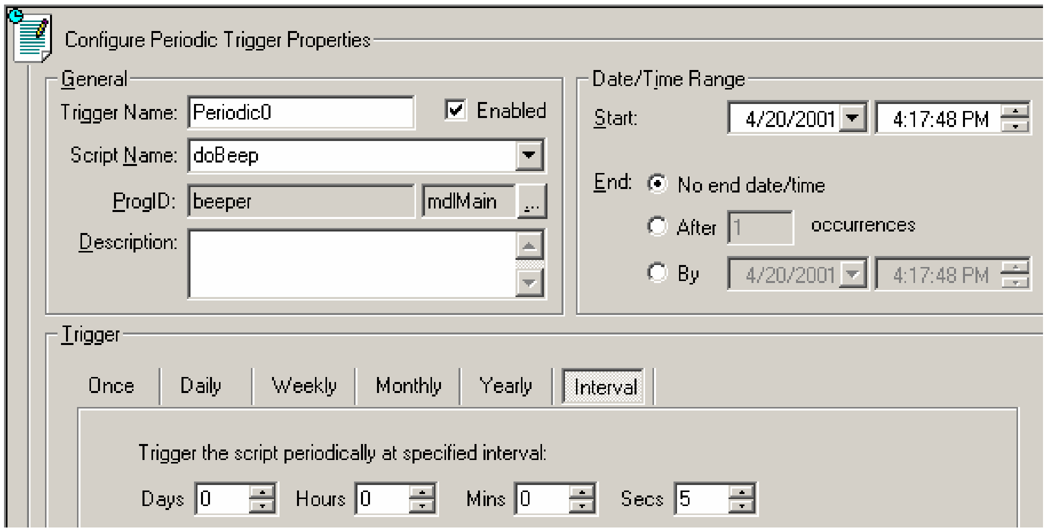
Example 2

1. Open a new ScriptWorX file in the user interface.

2. Insert a new periodic script as described above.

3. Enter the following information into the proper fields for this new script, as shown below.

Field	Entry
Script Name:	doBeep
Project:	beeper
Module/Designer:	mdlMain
Start Date/Time:	Current
End	No end date/time
Trigger Interval:	5 sec



Trigger Name: Periodic0 [✓ Enabled]
Script Name: doBeep
ProgID: beeper mdlMain
Description:
Start: 4/20/2001 4:17:48 PM
End: [✓ No end date/time] After 1 occurrences By 4/20/2001 4:17:48 PM
Trigger the script periodically at specified interval:
Days 0 Hours 0 Mins 0 Secs 5

Figure 37. doBeep Configuration

4. It is also necessary to set the working directory. This will be the directory where the compiled .dll file will be located, as should all configuration files directly related to this project. To set the working directory select **Set Working Directory** from the **Tools** menu.

5. Once the script has been configured as shown above, save the configuration file by selecting **Save As** from the **File** menu. Enter **Beeper.swx** as the file name.

6. Before this script is ready to be run, it is necessary to actually write the script. To do so, click the **Visual Basic Editor** button on the toolbar, or select **Macros - Visual Basic Editor** from the **Tools** menu.

7. In the Visual Basic Editor, open a new project by selecting **New Project** from the **File** menu. The **New Project** dialog box will open, as shown below.

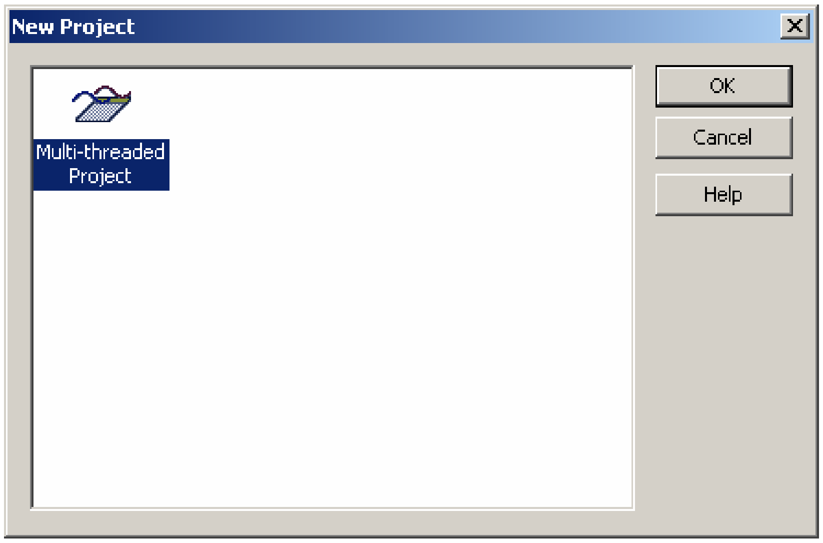


Figure 38. New Project Dialog Box in VBA Editor

8. Select **Multi-threaded Project** from the dialog box.

9. In the **Properties** window of the VBA Editor, change the name of the module from Designer1 to mdlMain. (**Note:** this is the same module name used in the configuration of the script.)

10. From the Project Explorer window, select the mdlMain object and enter the following code:

```
Public Sub doBeep()
   Beep
End Sub
```

Notice that the script name, doBeep, is used in the code. This is how the script will be able to run properly.

11. At this point the script is effectively written. It is now necessary to save and compile the project .dll and save the project. First save the project by selecting **Save As** from the **File** menu. Name the file **beeper.vba.** Make sure the file is saved into the working directory that you set earlier.

12. Now compile the project .dll by selecting **Make beeper.DLL** from the **File** menu.

13. Once the .dll has been compiled, close the Visual Basic Editor by selecting **Close and return to ScriptWorX** from the **File** menu.

14. Now it is possible to run the script from the user interface. To run the script, either select **Runtime!** from the menu bar, or click the **Runtime** button on the toolbar. This will launch the Runtime Monitor, which will show all scripts that are running, as well as each instance of each script that has run.

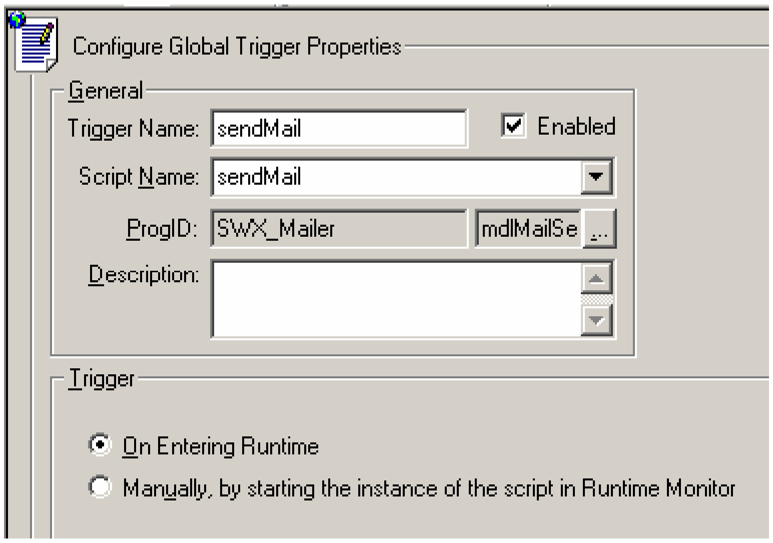
The script we have constructed in this example will beep every five seconds. If this does not occur during runtime, check your VBA code and your script configuration to make sure that the Script, Project, and Module names are consistent throughout.

Send Mail

This next example will start an instance of Microsoft Outlook, open a blank message file, address the message, and send the message.

1. To start off, insert a global script with the following properties.

Field	Entry
Script Name:	sendMail
Project:	SWX_Mailer
Module/Designer:	mdlMailSender
Start Date/Time:	Current
End	No end Date/Time
Trigger	On Entering Runtime


Configure Global Trigger Properties

General

Trigger Name: sendMail

[✓] Enabled

Script Name: sendMail

ProgID: SWX_Mailer mdlMailSe

Description:

Trigger

[● On Entering Runtime]

○ Manually, by starting the instance of the script in Runtime Monitor

Figure 39. Send Mail Script Configuration

2. Once you have set up the configuration according to the specified properties save the configuration by selecting **Save As** from the **File** menu. Enter **SWX_Mailer.swx** as the file name.

3. Now enter the code behind the script. To do so, start the Visual Basic Editor.

4. To use this example, you must have Microsoft Outlook installed on your computer, and the Outlook Object must be selected in the **Tools - References** menu in the VBA Editor.

5. Change the name of the module to **mdlMailSender,** and enter the following code in the code body window.

```
Public Sub sendMail()
  MsgBox "Before"
  Send_An_Email "test@smar.com", "", "", "myTest", "Hello from ScriptWorX"
  MsgBox "after"
End Sub


Sub Send_An_Email(P_to As String, P_cc As String, p_bcc As String, subject As String, m_text
As String)
```

```
On Error GoTo errHandler
Dim OutApp     As Outlook.Application
Dim OutMail    As Outlook.MailItem
Dim y          As Integer
Dim bodytext   As String   'text of e-mail
Dim Str_to_send As String 'value to send
  Set OutApp = CreateObject("Outlook.application")
  Set OutMail = OutApp.CreateItem(olMailItem)
  OutMail.To = P_to
  OutMail.CC = P_cc
  OutMail.BCC = p_bcc
  OutMail.subject = subject
  OutMail.Body = m_text
    OutMail.Send
    OutApp.Quit
      resetObjects:
  Set OutMail = Nothing
  Set OutApp = Nothing
  Exit Sub

errHandler:
    MsgBox Err.Description, vbCritical, Err.Number
    GoTo resetObjects
End Sub
```

6. Once the code is entered, save the code and compile the .dll.

7. After the .dll has been successfully created, close the Visual Basic Editor and return to ScriptWorX.

8. Now it is possible to run the script from the ScriptWorX user interface. To run the script, either select **Runtime!** from the menu bar, or click the **Runtime** button on the toolbar.

9. This will launch the Runtime Monitor, which will show all scripts that are running as well as each instance of each script that has run.

While this script is running, it will launch a message box with the label "Before" (this text can be change by replacing this entry in the code). Once the **OK** button is clicked in this box, the test message will be sent. After the message is sent, a similar message box with the label "After" will be displayed.

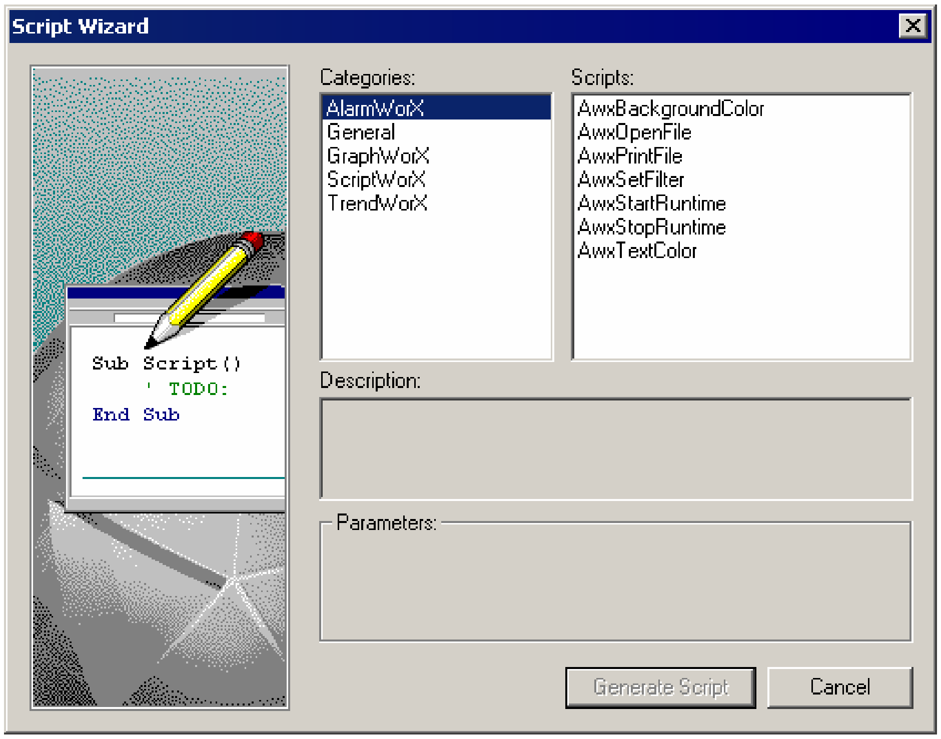
Note
To test to make sure this script is working correctly, enter your email address in place of test@smar.com.

Script Wizard

The **Script Wizard** is a tool that is available from each script configuration window and that allows you to generate scripts from script templates. Each script template is stored in one script template file (.stp), which is located in the Script Wizard Template directory placed in the installation directory of ScriptWorX. Script templates support several keywords, which are replaced during script generation by values entered by in the Wizard edit fields. Script templates also support the addition of necessary TypeLib references that can be appended to the VBA container. TypeLib references and optional keywords must be specified in the header section of the script.

Running the Script Wizard

To run the ScriptWorX VBA Script Wizard, insert a script configuration in the ScriptWorX user interface and click the **VBA Script Wizard** button on the configuration screen. Doing so will open the following window.



```
Sub Script()
    ' TODO:
End Sub
```

Figure 40. Script Wizard

The Script Wizard provides several categories of script templates from which to generate a script.

Description

The **Description** section tells you what the script is supposed to be doing and what components need to be installed on your system before the script will run properly. The text that is shown in this section is the text that was entered in the VBA code after the **#Description** keyword.

Parameters

Depending on the category and the script selected, several items are available in the **Parameters** section of the window. As stated in earlier sections, the parameters available are set in the configuration of the script template by using the **#Parameters** keyword. The entries for each of these fields will be directly inserted into the script and will determine exactly how the scripts run.

Installed Script Categories

There are five installed script template categories: AlarmWorX, General, GraphWorX, OPC Automation, and TrendWorX. Each of these categories includes associated scripts. Each script includes a description of what the script does and what components must be installed.

AlarmWorX

Script	Description
AwxBackgroundColor	Set background color of AWXView ActiveX. GraphWorX and AWXView must be installed.
AwxOpenFile	Open GraphWorX display with an AWXView ActiveX and load the AWXView configuration file.
AwxPrintFile	Print the display containing the AWXView ActiveX. GraphWorX and AWXView must be installed.
AwxSetFilter	Set filter name and filter expression for AWXView ActiveX. GraphWorX and AWXView must be installed.
AwxStartRuntime	Enter Runtime Mode. GraphWorX and AWXView must be installed.
AwxStopRuntime	Exit Runtime Mode. GraphWorX and AWXView must be installed.
AwxTextColor	Set text color of AWXView ActiveX. GraphWorX and AWXView must be installed.

General

Script	Description
Beeper	This script Beeps when started.
DiskSpace	The script obtains total and free disk space.
HelloSmar	The script pops up a message box that says "Hello from Smar."
Keypad	The script displays a Keypad form for entering a text string.
Memory	The script obtains total and available physical memory.
Numpad	The script displays a Numpad form for entering a double value.

GraphWorX

Script	Description
GwxArrayOfEllipses	Script starts GraphWorX and creates an array of ellipses, where the array dimensions and ellipse properties are specified.
GwxArrayOfRectangles	Script starts GraphWorX and creates an array of rectangles, where the array dimensions and rectangle properties are specified.
GwxClosePopup Window	Script starts GraphWorX and opens the popup window specified in the File Name field. Then the popup window is closed.
GwxEmbeddedWindow	Script starts GraphWorX and opens the display selected in

Script	Description
	the File Name field as an embedded window.
GwxIterateObjects	Iterates all symbols and their subsymbols in the display.
GwxMaximizeWindow	Script starts GraphWorX and maximizes its window. GraphWorX must be installed.
GwxMinimizeWindow	Script starts GraphWorX and minimizes its window. GraphWorX must be installed.
GwxOpenDisplay	Script starts GraphWorX and opens the display selected in the File Name field.
GwxPopupWindow	Script starts GraphWorX and opens a popup window specified in the File Name field. Use the Browse button.
GwxPrintDisplay	Script starts GraphWorX, and opens and prints the display selected in the File Name field.
GwxSetBackgroundColor	Script starts GraphWorX and changes the display background color.
GwxSetDisplayDimensions	Script starts GraphWorX and sets the display dimensions (work area/world bounds) of the currently loaded display.
GwxSetWindowDimensions	Script starts GraphWorX and sets the GraphWorX main window size and location as a percentage of the total screen size.
GwxStartRuntime	Script starts GraphWorX and switches to runtime mode. GraphWorX must be installed.
GwxStopRuntime	Script starts GraphWorX and stops runtime mode.
GwxToggleRuntime	Script toggles GraphWorX runtime mode. GraphWorX must be installed.

ScriptWorX

Script	Description
Infinite Script:	Implements body of an “infinite” script. It loops and waits for the ScriptWorX shut-down event.
Queue Source:	Queue source implementation with error handling.
Queue Target:	Queue target implementation with error handling – an infinite script with queue data reading.
ReadOPCTag:	Read an OPC tag using IDataPoint object; waits for the first update.
ReadOPCTag_NoWait:	The same as “ReadOPCTag” script, but it does not wait for the first update.
WriteOPCTag	Write an OPC tag using IDataPoint object.

TrendWorX

Script	Description
TwxAddTrend	Adds real-time trend to the TrendWorX Viewer. GraphWorX and TrendWorX Viewer must be installed.
TwxBackgroundColor	Sets TrendWorX Viewer background color. GraphWorX and TrendWorX Viewer must be installed.
TwxDeleteTrend	Deletes real-time trend pen. GraphWorX and TrendWorX Viewer must be installed.
TwxOpenFile	Script starts GraphWorX display with a TrendWorX Viewer ActiveX. GraphWorX and TrendWorX Viewer must be installed.
TwxPrintFile	Prints the display containing the TrendWorX Viewer ActiveX. GraphWorX and TrendWorX Viewer must be installed.
TwxShowDetails	Shows or hides the detailed list of trends. GraphWorX and TrendWorX Viewer must be installed.
TwxShowTitle	Shows or hides the TrendWorX Viewer title. GraphWorX and TrendWorX Viewer must be installed.
TwxStartRuntime	Enters runtime mode. GraphWorX and TrendWorX Viewer must be installed.
TwxStopRuntime	Exits runtime mode. GraphWorX and TrendWorX Viewer must be installed.

Using a Template

To access a script template, open the ScriptWorX user interface and click the **VBA Script Wizard** button. This will open the Script Wizard as described above.

1. Select the desired category and select the appropriate script from the list.

2. Click on the script. The description of the script should appear in the **Description** field.

3. Once you have set up the Wizard as desired, click the **Generate Script** button. The ScriptWorX user interface will be restored, and you will be asked if you want to view the VBA code.

Script Wizard Creation and Maintenance

This section is for more advanced users who wish to create their own script template files.

Header Section

The header section can contain optional keywords, which must be in the following format:

#KEYWORD: **value**

Optional keywords:

#REFERENCE:	*module1*
#REFERENCE:	*module2*
#REFERENCE:	**…**

– Add TypeLib references to VBA .

#DESCRIPTION:	*text1*
#DESCRIPTION:	*text2*
#DESCRIPTION:	**…**

- Enter script information that will be displayed in the Script Wizard dialog box.
- Description can be placed on several lines. Each of them must begin with the same keyword.

#PARAMETER: *parameter*

- *Parameter* may be one of the following strings.
- Pay attention to filling in this section; '***parameter***' must be spelled correctly!
- Parameters are exclusive; i.e. only one of them can be used at a time.

par_FileName	- Script template includes file name parameter.
par_Tag	- Script template includes OPC tag.
par_None	- Has no effect.

The header section must be ended by a keyword:

#END

This marks the end of header section. It is the only compulsory header keyword. The other keywords are optional.

Optional and Required Parameters in Scripts

Optional parameters must be defined in the header section as described above. These parameters are one of the following:

par_FileName, par_Tag

Required parameters can be used without definition in the header section (because the related entry fields are always used in the Script Wizard dialog box). Required parameters are:

par_Name, par_Key, par_Node

When parameters are used in the script template code, they must be enclosed by **<<** and **>>** characters.

Parameters are replaced during script generation by the values specified by the user in the related edit fields.

par_FileName	- When declared, the Script Wizard displays the edit field and **Browse** button to specify the file name.
par_Tag	- When declared, the Script Wizard displays the edit field and **Browse** button to specify the OPC tag.
par_Name	- Contains the script name.

par_Key	- GenRegistrar key parameter.
par_Node	- GenRegistrar node parameter.

Script Template Sample

```
#REFERENCE:       Gwx32
#DESCRIPTION:     Script starts GWX32 and opens display selected
#DESCRIPTION:     in the FileName field.
#DESCRIPTION:     Note Gwx32 must be installed.
#PARAMETER:       par_FileName
#END

' Script <<par_Name>> was generated by Smar Script Wizard
' from a template OpenDisplay.stp
'
Public Sub <<par_Name>>()
   ' Create an instance of GWX32
   Dim gwx As Gwx32.GwxDisplay
   Set gwx = New Gwx32.GwxDisplay

   If gwx Is Nothing then
      ' Report problem and exit
      MsgBox "GWX32 creation failed. Check it is installed and registered"
      Exit Sub
   End If

   call gwx.FileNew
   call gwx.FileOpen("<<par_FileName>>")
   call gwx.BringWindowToTop
   call gwx.ShowWindow

   MsgBox "Close GWX"
   call gwx.ExitApplication
   set gwx = nothing
End Sub
```

Example

In this example, we will create both a script wizard and a script as a result of this wizard, and will incorporate both the **#DESCRIPTION** and **#PARAMETER** keywords.

Building a Template

To start, open the **ProcessView/Bin/Script Wizard** directory. In this directory, you will find folders representing all of the above mentioned template categories.

1. Create a new category called **Tutorial** by creating a new folder in the Script Wizard directory and giving it the appropriate name.

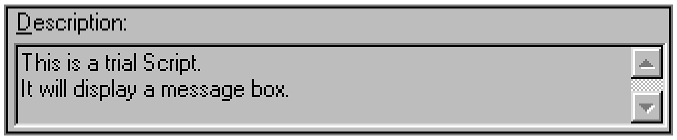
2. Create a new .txt file and rename it **Tutorial.stp.**

3. You have now created a blank file that can be edited using notepad. Double-click on this newly created file.

4. You are now ready to start writing the code for your Script Wizard. To enter the information that will appear in the **Description** field, enter the following code:

```
#DESCRIPTION: This is a trial script.
#DESCRIPTION: It will display a message box.
```

The reason for having two different lines is that the **Description** field in the Script Wizard interface has a preset length and does not wrap text within the box. Once the Wizard is saved, check to make sure that all of your description text is visible to the user, as shown below:



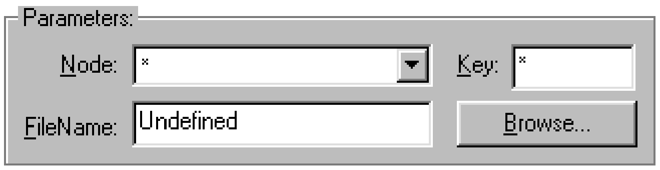
This is a trial Script.
It will display a message box.

Figure 41. Description Section

5. Enter the following parameter keyword:

```
#PARAMETER:   par_FileName
```

Entering this parameter will add the FileName field to the Script Wizard, as shown below:



Node: *
Key: *
FileName: Undefined

Figure 42. Parameter Section

Note
If the par_Tag parameter had been used, the FileName field would be attached to an OPC tag, and the **Browse** button would launch the OPC Universal Tag Browser.

6. Enter the end keyword: #END

This will indicate the end of the header section of the Wizard.

The code that makes the basis of the script is Visual Basic code, which references the different parameters set in the header. For our example, we will be creating a simple message box referencing a file selected by the user.

Note
The complexity of what a Script Wizard can accomplish is based solely on the extent to which the programmer is able to program in Visual Basic.

7. Enter the following code directly beneath the header section:

```
' Script <<par_Name>> (generated by the ScriptWizard from Tutorial)
Public Sub <<par_Name>>()
    MsgBox "What do you want to know about <<par_FileName>>!!!"
End Sub
```

The entry for <<par_Name>> is taken from the ScriptWorX user interface **Script Name** field. The entry for <<par_FileName>> is set in the Script Wizard user interface and is available since it was added in the header section.

After the script is coded as desired, save the file and close the instance of Notepad.

Using the Template

To access the script template that you have just created, open the ScriptWorX user interface and click the **VBA Script Wizard** button. This will open the Script Wizard as described above, but the Wizard should now contain an additional category called **Tutorial.**

Note
If this new category is not available, check to make sure that the folder containing the template is located in the same directory as the other template category folders.

1. Select the Tutorial category. The script template **Example** should be displayed in the scripts list.

2. Click on this script. The information you entered using the #Description keywords should appear in the **Description** field, and the **Parameters** section should contain the default **Node** and **Key** fields as well as the Optional **FileName** field.

3. To select a file name to be used in the script, click the **Browse** button and use the **Open** dialog to browse through available files.

4. Once you have set up the Wizard as desired, click the **Generate Script** button. This returns you to the ScriptWorX user interface. You will be asked if you want to view the VBA code.