Endpoint Security Scripting Reference

ZENworks® 11 Support Pack 2

July 2, 2012

Novell®

Legal Notices

Novell, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2007-2012 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide 5

1 Script Development 7

1.1 Scripting Languages . . . 7
1.2 Execution Context . . . 8
1.3 Event Triggers . . . 8
1.4 Namespaces . . . 8
1.5 Storage Interface . . . 9
1.5.1 Variables . . . 9
1.5.2 Temporary Storage Methods . . . 10
1.5.3 Persistent Storage Methods . . . 11
1.5.4 JScript Example . . . 12
1.5.5 VBScript Example . . . 12
1.6 Script Management Interface . . . 13
1.6.1 Script Information and Helper Methods . . . 13
1.6.2 Version Methods . . . 14
1.6.3 Trigger Event Methods . . . 15
1.6.4 Script Run Methods . . . 18
1.6.5 Program Launch/Execute Methods . . . 19
1.6.6 Display Methods . . . 21
1.6.7 Prompt Methods . . . 24
1.6.8 Safe Arrays . . . 26
1.6.9 Object Match Lists . . . 27
1.7 Effective Policy Interface . . . 28
1.7.1 PolicyInformation Object . . . 28
1.7.2 Effective Policies Methods . . . 29
1.8 Location Interface . . . 30
1.8.1 Definitions . . . 30
1.8.2 Data Types . . . 31
1.8.3 Security Location Methods . . . 32
1.8.4 Mobile (Unknown) Location Methods . . . 35
1.8.5 Assigned Location Methods . . . 35
1.8.6 Network Location Methods . . . 36
1.8.7 JScript Example . . . 36
1.8.8 VBScript Example . . . 37
1.9 Communication Hardware Policy Interface . . . 37
1.9.1 Data Types . . . 37
1.9.2 Enforced Policy Methods . . . 38
1.9.3 Hardware Enforcement Methods . . . 38
1.9.4 Adapter Connection Methods . . . 39
1.9.5 JScript Example . . . 40
1.9.6 VBScript Example . . . 40
1.10 WiFi Policy Interface . . . 41
1.10.1 Data Types . . . 41
1.10.2 Adhoc WiFi Networks Methods . . . 42
1.10.3 Block WiFi Connections . . . 43
1.10.4 Minimum Security Level Methods . . . 44
1.10.5 Minimum Signal Strength Methods . . . 44
1.11 Storage Device Control Policy Interface . . . 45
1.11.1 Data Types . . . 46
1.11.2 AutoPlay Methods . . . 46

1.11.3 Volumes Methods 47

2 Script Testing 49

2.1 Enabling Script Testing in the Endpoint Security Agent 49
2.2 Testing an Unpublished Script 49
2.3 Testing a Published Scripting Policy 53
2.4 Tracing a Script's Execution 55

About This Guide

This *Novell ZENworks 11 Endpoint Security Scripting Reference* provides information to help you create and test scripts to be used in Scripting policies.

The information in this guide is organized as follows:

- Chapter 1, “Script Development,” on page 7
- Chapter 2, “Script Testing,” on page 49

Audience

This guide is written for the ZENworks Endpoint Security Management administrators.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation.

Additional Documentation

ZENworks Endpoint Security Management is supported by other documentation (in both PDF and HTML formats) that you can use to learn about and implement the product. For additional documentation, see the ZENworks 11 documentation Web site (http://www.novell.com/documentation/zenworks11).

1 Script Development

The following sections provide information to help you develop the script content for use in Scripting policies

- Section 1.1, “Scripting Languages,” on page 7
- Section 1.2, “Execution Context,” on page 8
- Section 1.3, “Event Triggers,” on page 8
- Section 1.4, “Namespaces,” on page 8
- Section 1.5, “Storage Interface,” on page 9
- Section 1.6, “Script Management Interface,” on page 13
- Section 1.7, “Effective Policy Interface,” on page 28
- Section 1.8, “Location Interface,” on page 30
- Section 1.9, “Communication Hardware Policy Interface,” on page 37
- Section 1.10, “WiFi Policy Interface,” on page 41
- Section 1.11, “Storage Device Control Policy Interface,” on page 45

1.1 Scripting Languages

The Endpoint Security Agent uses the Microsoft Windows Script Host (WSH) to run scripts on a device. All scripts are subject to WSH restrictions. Script content can be authored in either JScript or VBScript language; using multiple languages (JScript and VBScript together) in the same script is not supported.

Standard WSH coding methods are supported, with the following exceptions:

1. WScript.Echo is not supported because return values can’t be sent back to a parent window that is unavailable. Use the Action.DisplayMessage methods instead (see Section 1.6.6, “Display Methods,” on page 21).
2. Access Shell Objects. Use the following modified nomenclature/call:

```
[JScript]
   Use:
   var WshShell = new ActiveXObject("WScript.Shell");
   Instead of:
   var WshShell = WScript.CreateObject ("WScript.Shell");

[VBScript]
   Use:
   Dim WshShell
   Set WshShell = CreateObject("WScript.Shell")
   Instead of:
   Dim WshShell
   Set WshShell = WScript.CreateObject("WScript.Shell")
```

1.2 Execution Context

Scripts execute in either the System context or the User context. The execution context is defined in the Scripting policy through the *Run As* setting.

The script context, along with the operating system, determines the rights provided to the script and the functions it can perform. For example:

- On Windows Vista and Windows 7 operating systems, a script running in the System context (Session 0) cannot display messages on it own. To display messages, the script must use the Action.DisplayMessage methods or another mechanism. However, on Windows XP, scripts might be able to perform direct dialog displays.
- Scripts running in the User context execute with the right of the user session.
- Scripts running in the System context have the same rights as Windows services.

1.3 Event Triggers

Triggers are events that cause the Endpoint Security Agent to determine when and if a script should be executed. These events can either be internal agent events or external events monitored by the agent. A script is run when one of the triggers is fired, the script is not already running, and the scripting context (system or user) is available.

Triggers are defined in the Scripting policy. You cannot use a script to change the triggers, but you can use a script to discover the trigger that initiated a script. For information, see Section 1.6.3, "Trigger Event Methods," on page 15.

A brief description of each trigger is provided below.

- **Immediate:** Executes the script immediately on load of the script.
- **Location Change:** Executes the script when entering or leaving a location. Trigger can be applied to all location changes or specific location changes only.
- **Network Change:** Executes the script when a network environment that is used for location determination changes, even if the network change does not cause a location change.
- **Network Connect:** Executes the script when the wired adapter, wireless adapter, or modem detects a new connection.
- **Network Disconnect:** Executes the script with the wired adapter, wireless adapter, or modem loses a connection.
- **Policy Change:** Executes the script when the effective policy is updated.
- **Timer:** Executes the script every *n* minutes after the initial enforcement of the policy. The interval includes a one-minute boundary, meaning that the script is run within a minute (plus or minus) of the end of the interval.

1.4 Namespaces

The Endpoint Security Agent provides three namespaces for a script to allow it to control or access the agent. The namespaces are as follows:

- **Query:** Provides methods to get the current state of the agent. For example, Query methods could provide information about the device's network environment, security location, and enforced policies.

- **Action:** Provides methods to change the behavior of the agent or interact with the user. For example, Action methods could display a message or message prompt, start or stop another script, or change the security location.
- **Storage:** Provides methods for the script to store variables for the current session (temporary) or across sessions (persistent) For example, stored variables could be used to hold the last execution time or to transfer data between script executions.

All methods begin with one of the three namespaces. For example:

- string Query.ScriptName
- int Action.TriggerScript(string *script*, string *reason*)
- string Storage.GetNameValue(string *name*)

1.5 Storage Interface

The Storage interface provides a way to save variable data. Variables can be saved in temporary storage or persistent storage.

- Section 1.5.1, “Variables,” on page 9
- Section 1.5.2, “Temporary Storage Methods,” on page 10
- Section 1.5.3, “Persistent Storage Methods,” on page 11
- Section 1.5.4, “JScript Example,” on page 12
- Section 1.5.5, “VBScript Example,” on page 12

1.5.1 Variables

Scripting variables can be used to store information for use in the current Endpoint Security Agent session (temporary variables) or for use across sessions (persistent variables).

As you use variables, be aware of the following naming conventions:

- Variable names can contain any printable character.
- Variable names are not explicitly limited in size.
- A global variable is defined by prepending a forward slash (/) to the variable name. Global variables are available to other scripts. For example: Storage.NameValueExists(“/boolWarnedOnPreviousLoop”).
- Any variable that does not start with a forward slash (/) is a local variable. Local variables are available only to the script that created them.
- Variables are stored in either temporary storage or persistent storage (for details, see Section 1.5, “Storage Interface,” on page 9). Variable names are unique to each storage system. If a script uses the same name for a variable in both the temporary and persistent storage, the values are independent of each other despite the name being the same.

1.5.2 Temporary Storage Methods

Temporary storage allows a variable to be retained for the current Endpoint Security Agent session only. The variable is lost when the agent shuts down.

All variables are considered local to the script unless the variable name follows the naming conventions for a global variable. Local variables use the script's identifier to ensure uniqueness. If the script identifier is changed, the script no longer has access to its local variables.

bool Storage.NameValueExists(string *name*)

Description:	Determines if a temporary variable already exists.
Parameters:	*name* — variable name being requested
Returns:	`True` if the variable is found in the store. `False` if not.

string Storage.GetNameValue(string *name*)

Description:	Gets the value associated with a temporary variable.
Parameters:	*name* — variable name being requested
Returns:	The value being stored. If the value does not exist, an empty string is returned.

int Storage.SetNameValue(string *name*, string *value*)

Description:	Sets the value for a temporary variable.
Parameters:	*name* — variable name in which to store the value
	value — value to store
Returns:	`0` on success. Any other number on failure.

int Storage.ClearNameValue(string *name*)

Description:	Clears the value for a temporary variable.
Parameters:	*name* — name of variable to clear
Returns:	`0` on success. Any other number on failure.

1.5.3 Persistent Storage Methods

Persistent storage allows a variable to be retained across Endpoint Security Agent restarts; the variable can only be cleared by script or by using the Agent Status feature in the Endpoint Security Agent's About box.

All variables are considered local to the script unless the variable name follows the naming conventions for a global variable. Local variables use the script's identifier to ensure uniqueness. If the script identifier is changed, the script no longer has access to its local variables.

bool Storage.PersistValueExists(string *name*)

Description:	Determines if a persistent variable already exists.
Parameters:	*name* — variable name being requested
Returns:	True if the variable is found in the store. False if not.

string Storage.GetPersistValue(string *name*)

Description:	Gets the value associated with a persistent variable.
Parameters:	*name* — variable name being requested
Returns:	The value being stored. If the value does not exist, an empty string is returned.

int Storage.SetPersistValue(string *name*, string *value*)

Description:	Sets the value for a persistent variable.
Parameters:	*name* — variable name in which to store the value
	value — value to store
Returns:	0 on success. Any other number on failure.

int Storage.ClearPersistValue(string *name*)

Description:	Clears the value for a persistent variable.
Parameters:	*name* — name of variable to clear
Returns:	0 on success. Any other number on failure.

1.5.4 JScript Example

```
var ret;
var curValue = 0;
if (Storage.NameValueExists("testval"))
    curValue = Storage.GetNameValue("testval");
curValue++;
ret = Storage.SetNameValue("testval", curValue);
Action.Trace("NameValue = " + curValue);
Action.DisplayMessage("Storage", "Name Value: " + curValue, "Info", 3);
Action.Sleep(3000);

curValue = 0;
if (Storage.NameValueExists("/testval"))
    curValue = Storage.GetNameValue("/testval");
curValue++;
ret = Storage.SetNameValue("/testval", curValue);
Action.Trace("Shared NameValue = " + curValue);
Action.DisplayMessage("Shared Storage", "Name Value: " + curValue, "Info", 3);
Action.Sleep(3000);

curValue = 0;
if (Storage.PersistStringExists("testval"))
    curValue = Storage.GetPersistString("testval");
curValue++;
ret = Storage.SetPersistString("testval", curValue);
Action.Trace("Persist String = " + curValue);
Action.DisplayMessage("Storage", "Persist String: " + curValue, "Info", 3);
Action.Sleep(3000);

curValue = 0;
if (Storage.PersistStringExists("/testval"))
    curValue = Storage.GetPersistString("/testval");
curValue++;
ret = Storage.SetPersistString("/testval", curValue);
Action.Trace("Shared Prersist String = " + curValue);
Action.DisplayMessage("Shared Storage", "Persist String: " + curValue, "Info", 3);
Action.Sleep(3000);
```

1.5.5 VBScript Example

```
dim ret
dim curValue
curValue = 0

If Storage.NameValueExists("testval") then
    curValue = Storage.GetNameValue("testval")
End If
curValue = curValue + 1
ret = Storage.SetNameValue("testval", curValue)
Action.Trace "NameValue = " & curValue
msg = "Name Value: " & curValue

Action.DisplayMessage "Storage", msg, "Info", 3
Action.Sleep 3000

curValue = 0
If Storage.NameValueExists("/testval") then
    curValue = Storage.GetNameValue("/testval")
End If

curValue = curValue + 1
ret = Storage.SetNameValue("/testval", curValue)
Action.Trace "Shared NameValue = " & curValue
Action.DisplayMessage "Shared Storage", "Name Value: " & curValue, "Info", 3
Action.Sleep 3000
```

```
curValue = 0
If Storage.PersistStringExists("testval") then
    curValue = Storage.GetPersistString("testval")
End If
curValue = curValue + 1
ret = Storage.SetPersistString("testval", curValue)
Action.Trace "Persist String = " & curValue
Action.DisplayMessage "Storage", "Persist String: " & curValue, "Info", 3
Action.Sleep 3000

curValue = 0
If Storage.PersistStringExists("/testval") then
    curValue = Storage.GetPersistString("/testval")
End If
curValue = curValue + 1
ret = Storage.SetPersistString("/testval", curValue)
Action.Trace "Shared Prersist String = " & curValue
Action.DisplayMessage "Shared Storage", "Persist String: " & curValue, "Info", 3
Action.Sleep 3000
```

1.6 Script Management Interface

The Script Management interface provides methods for getting script information, launching other scripts and programs, and displaying informational messages and prompts to users. The methods are organized into the following sections:

- Section 1.6.1, “Script Information and Helper Methods,” on page 13
- Section 1.6.2, “Version Methods,” on page 14
- Section 1.6.3, “Trigger Event Methods,” on page 15
- Section 1.6.4, “Script Run Methods,” on page 18
- Section 1.6.5, “Program Launch/Execute Methods,” on page 19
- Section 1.6.6, “Display Methods,” on page 21
- Section 1.6.7, “Prompt Methods,” on page 24
- Section 1.6.8, “Safe Arrays,” on page 26
- Section 1.6.9, “Object Match Lists,” on page 27

1.6.1 Script Information and Helper Methods

The Script Information and Helper methods get information about a script (name, ID, and execution context) and provide general script helping functions such as creating a new unique ID for use in the script, generating trace messages for the script, and pausing the script for a specified amount of time.

string Query.ScriptName

Description: Gets the name of the script. The name is derived from the Scripting policy name.

string Query.ScriptId

Description: Gets the script identifier. The identifier is derived from the Scripting policy ID.

string Query.ScriptContext

Description: Gets the context (user or system) in which the script is running.

string Query.UniqueID

Description: Generates a unique identifier for use by the script.

void Action.Trace(string *msg*)

Description: Sends trace messages to the user or service logs (depending on whether the script is running in the user context or system context). Each trace message has its script id concatenated to the message.

The trace messages can also be viewed in the Script Tracing dialog of the Endpoint Security Agent About box.

Parameters: *msg* — The message string to log.

void Action.Sleep (int *millisec*)

Description: Causes the script to sleep for a specified period of time.

Parameters: *millisec* — The number of milliseconds the script sleeps before proceeding. The implementation wakes up on a regular interval to check if the script needs to be terminated early due to a policy change or agent restart. Control is returned only after the number of milliseconds has expired.

JScript Example

```
Action.Trace("");
Action.Trace(" ******** Script Information ********* ");
Action.Trace("UniqueID: " + Query.UniqueID);
Action.Trace("Script Name: " + Query.ScriptName);
Action.Trace("Script ID: " + Query.ScriptID);
Action.Trace("Script Context: " + Query.ScriptContext);
```

VBScript Example

```
Action.Trace ""
Action.Trace " ******** Script Information ********* "
Action.Trace "UniqueID: " & Query.UniqueID
Action.Trace "Script Name: " & Query.ScriptName
Action.Trace "Script ID: " & Query.ScriptID
Action.Trace "Script Context: " & Query.ScriptContext
```

1.6.2 Version Methods

The Version methods get information about the version of a namespace (Query, Action, Storage) or of the Endpoint Security Agent.

int Query.Version(string *category*, string *component*)

Description: Gets the version of the specified namespace or of the Endpoint Security Agent.

Parameters: *category* — One of the following four identifiers: `query`, `action`, `storage`, `client`.

component — The requested version component. The four identifiers are: `major`, `minor`, `revision`, `build`.

Returns: An integer value for the requested component. If an invalid component is requested, -1 is returned.

JScript Example

```
Action.Trace("");
Action.Trace(" ******** Version Information ********* ");
Action.Trace("");
Action.Trace("Client: " + Query.Version("Client", "Major") + "." + Query.Version("
Client", "Minor") + "." + Query.Version("Client", "Revision") + "." + Query.Versio
n("Client", "Build"));
Action.Trace("Query: " + Query.Version("Query", "Major") + "." + Query.Version("Qu
ery", "Minor") + "." + Query.Version("Query", "Revision") + "." + Query.Version("Q
uery", "Build"));
Action.Trace("Action: " + Query.Version("Action", "Major") + "." + Query.Version("
Action", "Minor") + "." + Query.Version("Action", "Revision") + "." + Query.Versio
n("Action", "Build"));
Action.Trace("Storage: " + Query.Version("Storage", "Major") + "." + Query.Version
("Storage", "Minor") + "." + Query.Version("Storage", "Revision") + "." + Query.Ve
rsion("Storage", "Build"));
```

VBScript Example

```
Function DisplayVersion (name)
    dim major
    dim minor
    dim revision
    dim build

    major = Query.Version(name, "Major")
    minor = Query.Version(name, "Minor")
    revision = Query.Version(name, "Revision")
    build = Query.Version(name, "Build")
    Action.Trace name & ": " & major & "." & minor & "." & revision & "." & build
End Function

Action.Trace ""
Action.Trace " ******** Version Information ********* "
Action.Trace ""
DisplayVersion("Client")
DisplayVersion("Query")
DisplayVersion("Action")
DisplayVersion("Storage")
```

1.6.3 Trigger Event Methods

The Trigger Event methods get information about the event that caused the script to execute.

Trigger Reasons

The following table lists the reasons a script is triggered. Each trigger reason includes one or more indexes that are available for the trigger. The indexes listed for each trigger are guaranteed to be available. Other indexes, and even other reasons, might be available depending on the version of the Endpoint Security Agent.

Trigger Reason	Index	Description
Location change	reason	The trigger reason value. For a location change, the value is always `location`.
	switch_from_id	The ID of the switched-from location.
	switch_from	The name of the switched-from location.
	switch_to_id	The ID of the switched-to location.
	switch_to	The name of the switched-to location
	change_reason	Reason for the location change that triggered the script; for reasons, see Section 1.8.2, “Data Types,” on page 31
Network environment change	reason	The trigger reason value. For a network environment change, the value is always `network_environment`.
Network connect	reason	The trigger reason value. For a network connection, the value is always `network_connect`.
	device_id	The device ID of the adapter that detected the connection
Network disconnect	reason	The trigger reason value. For a network disconnection, the value is always `network_disconnect`.
	device_id	The device ID of the adapter that detected the disconnect
Immediate	reason	The trigger reason value. For an immediate trigger, the value is always `immediate`.
	caller	(Optional) The name of the script that initiated the trigger.
	caller_ID	(Optional) The ID of the script that initiated the trigger,
	caller_reason	(Optional) The reason the script initiated the trigger.
Timer	reason	The trigger reason value. For a time trigger, the value is always `timer`.
	interval	The time interval (in minutes) that triggered the script

string Query.TriggerParameter(string *index*)

Description: Gets the value of the requested index.

Parameters: *index* — One of the index names listed in "Trigger Reasons" on page 16. For example, `location` or `switch_from`.

Returns: The value of the requested index value. For example, if `reason` is the index, the value might be `location` or `network_connect`. If `switch_from` is the index, the value might be `work` or `office`.

If an index is out of range or invalid, an empty string is returned.

int Query.TriggerParameterCount

Description: Gets the number of indexes for the trigger. For example, if Location change is the trigger, 6 or more indexes can be available.

Returns: The number of indexes.

string Query.TriggerParameterName(int *index*)

Description: Gets the name of the requested index.

Parameters: *index* — The number of the index being requested. Index names are listed in "Trigger Reasons" on page 16. Index numbers are not listed because they can change from one script run to another. For example, the `reason` index might be 0 during one run and 4 during another.

Returns: The name of the requested index. For example, `switch_from_ID`, `deviceID`, or `reason`.

string Query.TriggerParameterValue(int *index*)

Description Gets the value of the requested index.

Parameters: *index* — The number of the index being requested. Index names are listed in "Trigger Reasons" on page 16. Index numbers are not listed because they can change from one script run to another. For example, the `reason` index might be 0 during one run and 4 during another.

Returns: The value of the requested index. For example, if `switch_from` is the requested index (based on its index number, not name), the value might be `work` or `office`.

JScript Example

```
Action.Trace("");
Action.Trace(" ******** Trigger Reasons ********* ");
Action.Trace("");
Action.Trace("Reason = " + Query.TriggerParameter("reason"));
Action.Trace("Parameter Count = " + Query.TriggerParameterCount);
for(var idx = 0; idx < Query.TriggerParameterCount; idx++)
{
    Action.Trace("Parameter: " + Query.TriggerParameterName(idx) + " -
> " + Query.TriggerParameterValue(idx));
}

Action.Trace("Invalid trigger parm return: " + Query.TriggerParameter("-1"));
```

VBScript Example

```
Action.Trace ""
Action.Trace " ******** Trigger Reasons ********* "
Action.Trace ""
Action.Trace "Reason = " & Query.TriggerParameter("reason")
Action.Trace "Parameter Count = " & Query.TriggerParameterCount
For idx = 0 to (Query.TriggerParameterCount - 1)
    Action.Trace "Parameter: " & Query.TriggerParameterName(idx) & " -
> " & Query.TriggerParameterValue(idx)
Next

Action.Trace "Invalid trigger parm return: " & Query.TriggerParameter("-1")
```

1.6.4 Script Run Methods

The Script Run methods trigger or terminate another script in the system.

int Action.TriggerScript(string *script*, string *reason*)

Descript ion: Triggers another script in the system.

Parameters: *script* — The name or ID of the script being requested to run.

reason — Passed along as part of the trigger parameter. The script that is called has the value stored as the *caller_reason* trigger parameter.

Returns: The following are common return values. Other values are also possible:

- `0` — The script was found and the trigger will be attempted.
- `50` — The action is not supported; could be returned because the script is attempting to trigger itself.
- `1168` — The script was not found in the system.
- `Other non-zero values` — The script failed to run.

int Action.TerminateScript(string *script*, string *reason*)

Description: Terminates another script in the system by name or id. This does not unload the script.

Parameters: *script* — The name or ID of the script being requested to run.

reason — Passed along as part of the trigger parameter. The script that is called has the value stored as the *caller_reason* trigger parameter.

Returns: The following are common return values. Other values are also possible:

- `0` — The script was found and the trigger will be attempted.
- `50` — The action is not supported; could be returned because the script is attempting to terminate itself.
- `1168` — The script was not found in the system.
- `Other non-zero values` — The script failed to run.

1.6.5 Program Launch/Execute Methods

The Launch/Execute methods provide ways to launch and execute programs. A launch method runs the program but does not wait for the program to finish and return an exit code. An execute method runs the program and waits for it to finish and return an exit code, or for the execution timeout to expire.

A launched or executed program runs in the same context (user or system) as the script, unless the script overrides the context by passing a new context.

Be aware that Windows Vista, Windows 7 and Windows Server 2008 do not allow GUI applications to display in the system context.

int Action.Launch(string *context*, bool *hide*, string *command*, string *parameters*)

Description: Starts a program in the requested context. The script continues without waiting for the program to return an exit code.

Parameters: *context* — Valid inputs are user or system. Leave the parameter empty to run the program in the same context as the script. If the user context is requested and the primary user context is unavailable, an error code is returned and the request is dropped.

hide — If `true`, the command shell used to launch the program is not displayed. If `false`, the command shell is displayed.

command — The command to execute. If the command starts with `http:` or `www.`, the link is launched using the default web browser.

parameters — Parameters to be passed to the command.

Returns: The following are common return values. Other values are also possible:

- `0` — Success
- `31` — General failure. The launching of the program failed due to a file not found, the command failing, or other similar reason.
- `1359` — The launch context (user or system) is not available.

int Action.Execute(string *context*, bool *hide*, string *command*, string *parameters*)

Description: Starts a program in the requested context. The script pauses until the program returns an exit code.

Parameters: *context* — Valid inputs are user or system. Leave the parameter empty to run the program in the same context as the script. If the user context is requested and the primary user context is unavailable, an error code is returned and the request is dropped.

hide — If true, the command shell used to execute the program is not displayed. If false, the command shell is displayed.

command — The command to execute. If the command starts with `http:` or `www.`, the link is launched using the default web browser.

parameters — Parameters to be passed to the command.

Returns: In addition to the exit code of the executed program, the following errors can be returned:

- `31` — General failure. Execution failed due to a file not found, the command failing, or other similar reasons.
- `1359` — The execute context (user or system) is not available.

int Action.ExecuteWithTimeout(string *context*, bool *hide*, string *command*, string *parameters* int *timeout*)

Description: Starts a program in the requested context. The script pauses until the program returns an exit code or until the timeout is reached.

Parameters: *context* — Valid inputs are user or system. Leave the parameter empty to run the program in the same context as the script. If the user context is requested and the primary user context is unavailable, an error code is returned and the request is dropped.

hide — If true, the command shell used to execute the program is not displayed. If false, the command shell is displayed.

command — The command to execute. If the command starts with `http:` or `www.`, the link is launched using the default web browser.

parameters — Parameters to be passed to the command.

timeout — Number of seconds to wait for an exit code from the program.

Returns: In addition to the exit code of the executed program, the following errors can be returned:

- `31` — General failure. Execution failed due to a file not found, the command failing, or other similar reasons.
- `121` — The command was successfully executed but did not complete before the timeout was reached.
- `1359` — The execute context (user or system) is not available.

JScript Example

```
var ret;

ret = Action.Launch("user", false, "notepad", "");
Action.Trace("User: Launch notepad: " + ret);

ret = Action.Execute("user", false, "notepad", "");
Action.Trace("User: Execute notepad: " + ret);

ret = Action.ExecuteWithTimeout("user", false, "notepad", "", 5);
Action.Trace("User: Execute with Timeout, notepad: " + ret);
```

VBScript Example

```
dim ret

ret = Action.Launch("user", false, "notepad", "")
Action.Trace("User: Launch notepad: " & ret)

ret = Action.Execute("user", false, "notepad", "")
Action.Trace("User: Execute notepad: " & ret)

ret = Action.ExecuteWithTimeout("user", false, "notepad", "", 5)
Action.Trace("User: Execute with Timeout, notepad: " & ret)
```

1.6.6 Display Methods

The Display methods enable a message to be displayed to a user. The methods are valid only if the script is running in a user session.

The displayed message includes an OK button to dismiss the message. You can also set a timeout to automatically dismiss the message. The message does not pause the script; it continues to run while the message displays.

Display messages are intended for providing information to the user. If you need to display a message that requires the user to make a choice (such as OK or Cancel), you should use a message prompt. See Section 1.6.7, "Prompt Methods," on page 24.

void Action.DisplayMessage(string *title*, string *message*, string *icon*, int *timeout*)

Description: If a primary user process is running, displays a custom message to the user. If no primary user process is a vial able, the message is dropped.

Parameters: *title* — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), default. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

void Action.DisplayMessageWithLink(string *title*, string *message*, string *icon*, int *timeout*, string *linkName*, string *linkCommand*, string *linkParameters*)

Description: If a primary user process is running, displays a custom message to the user. If no primary user process is available, the message is dropped.

Parameters: *title* — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), `default`. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

linkName — The name of the link to be display on the dialog box.

linkCommand — The command to be executed when the link is clicked.

linkParameters — Parameters to be passed as part of the execution command.

void Action.DisplayMessageById(string *id*, string *title*, string *message*, string *icon*, int *timeout*)

Description: If a primary user process is running, displays a custom message to the user. If no primary user process is available, the message is dropped.

Parameters: *id* — Provides that ability for message suppression. If a message with the same id is already being displayed to the user, this message is dropped.

title — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), `default`. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

void Action.DisplayMessageByIdWithLink(string *id*, string *title*, string *message*, string *icon*, int *timeout*, string *linkName*, string *linkCommand*, string *linkParameters*)

Description: If a primary user process is running, displays a custom message to the user. If no primary user process is available, the message is dropped.

Parameters: *id* — Provides that ability for message suppression. If a message with the same id is already being displayed to the user, this message is dropped.

title — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), `default`. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

linkName — The name of the link to be display on the dialog box.

linkCommand — The command to be executed when the link is clicked.

linkParameters — Parameters to be passed as part of the execution command.

JScript Example

```
Action.DisplayMessage("Display Message", "Error icon", "Error", 2);
Action.Sleep(2000);

Action.DisplayMessageWithLink("Display Message With Link", "Error icon", "Error",
2, "novell", "www.novell.com", "");
Action.Sleep(2000);

Action.DisplayMessageById("2", "Display Message By Id", "Should See", "app", 5);
Action.Sleep(2000);
Action.DisplayMessageById("2", "Display Message By Id", "Should not see", "error",
 2);
Action.Sleep(3000);

Action.DisplayMessageByIdWithLink("8", "Display Message By Id With Link", "Should
See", "app", 5, "novell", "www.novell.com", "");
Action.Sleep(2000);
Action.DisplayMessageByIdWithLink("8", "Display Message By Id With Link", "Should
not see", "error", 2, "novell", "www.novell.com", "");
```

VBScript Example

```
Action.DisplayMessage "Display Message", "Error icon", "Error", 2
Action.Sleep 2000

Action.DisplayMessageWithLink "Display Message With Link", "Error icon", "Error",
2, "novell", "www.novell.com", ""
Action.Sleep 2000

Action.DisplayMessageById "2", "Display Message By Id", "Should See", "app", 5
Action.Sleep 2000
Action.DisplayMessageById "2", "Display Message By Id", "Should not see", "error",
 2
Action.Sleep 3000

Action.DisplayMessageByIdWithLink "8", "Display Message By Id With Link", "Should
See", "app", 5, "novell", "www.novell.com", ""
Action.Sleep 2000
Action.DisplayMessageByIdWithLink "8", "Display Message By Id With Link", "Should
not see", "error", 2, "novell", "www.novell.com", ""
```

1.6.7 Prompt Methods

The Prompt methods enable a message prompt to be displayed to a user. The methods are valid only if the script is running in a user session.

The prompt can include different response buttons, such as OK/Cancel or Abort/Retry/Ignore. You can also set a timeout to automatically close the prompt if the user doesn't respond.

Message prompts are intended for prompting the user to make a choice. If you only need to display information to the user, you should use a display message. See Section 1.6.6, "Display Methods," on page 21.

string Action.Prompt(string *title*, string *message*, string *icon*, int *timeout*, string *buttons*)

Description: If a primary user process is running, displays a custom message prompt to the user. If no primary user process is available, the message prompt is dropped.

Parameters:

title — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), `default`. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

buttons — The buttons to display. Valid inputs are: `ok`, `okCancel`, `abortRetryIgnore`, `yesNoCancel`, `yesNo`, `retryCancel`. Inputs are not case-sensitive.

Returns: One of the following:

- "" — Empty string. The primary process is unavailable, no input received.
- `closed` — Dialog box closed without input.
- `timeout` — Dialog box timed out.
- `ok` — OK button selected.
- `cancel` — Cancel button selected.
- `abort` — Abort button selected.
- `retry` — Retry button selected.
- `ignore` — Ignore button selected.
- `yes` — Yes button selected.
- `no` — No button selected.
- `cancel` — Cancel button selected.

string Action.PromptWithLink(string *title*, string *message*, string *icon*, int *timeout*, string *buttons*, string *linkName*, string *linkCommand*, string *linkParameters*)

Description: If a primary user process is running, displays a custom message prompt to the user. If no primary user process is available, the message prompt is dropped.

Parameters: *title* — String displayed in the title bar.

message — The main message.

icon — The icon to display with the message. You can specify any of the following system icons or leave the string empty for no icon: `error`, `app`, `hand`, `info`, `quest`, `warn`, `exclamation` (or `!`), `stop`, `asterisk` (or `*`), `default`. Be aware that it is possible for no default system icon to exist.

timeout — The number of seconds for the message to display. Use 0 to display the message until the user closes the dialog box.

buttons — The buttons to display. Valid inputs are: `ok`, `okCancel`, `abortRetryIgnore`, `yesNoCancel`, `yesNo`, `retryCancel`. Inputs are not case-sensitive.

linkName — The name of the link to be display on the dialog box.

linkCommand — The command to be executed when the link is clicked.

linkParameters — Parameters to be passed as part of the execution command.

Returns: One of the following:

- "" — Empty string. The primary process is unavailable, no input received.
- `closed` — Dialog box closed without input.
- `timeout` — Dialog box timed out.
- `ok` — OK button selected.
- `cancel` — Cancel button selected.
- `abort` — Abort button selected.
- `retry` — Retry button selected.
- `ignore` — Ignore button selected.
- `yes` — Yes button selected.
- `no` — No button selected.
- `cancel` — Cancel button selected.

JScript Example

```
var ret;
ret = Action.Prompt("Prompt - Ok", "Hit ok", "Error", 0, "ok");
Action.Trace("Ok Result: " + ret);
ret = Action.Prompt("Prompt - OkCancel", "Hit ok", "", 0, "okCancel");
Action.Trace("Ok Result: " + ret);
ret = Action.Prompt("Prompt - Retry/
Cancel", "Allow to timeout", "", 5, "retryCancel");
Action.Trace("timeout Result: " + ret);

ret = Action.PromptWithLink("Prompt - Retry/
Cancel", "With link", "", 3, "retryCancel", "Novell", "www.novell.com", "");
Action.Trace("with link results: " + ret);
```

VBScript Example

```
dim ret
ret = Action.Prompt("Prompt - Ok", "Hit ok", "Error", 0, "ok")
Action.Trace("Ok Result: " & ret)
ret = Action.Prompt("Prompt - OkCancel", "Hit ok", "", 0, "okCancel")
Action.Trace("Ok Result: " & ret)
ret = Action.Prompt("Prompt - Retry/
Cancel", "Allow to timeout", "", 5, "retryCancel")
Action.Trace("timeout Result: " & ret)

ret = Action.PromptWithLink("Prompt - Retry/
Cancel", "With link", "", 3, "retryCancel", "Novell", "www.novell.com", "")
Action.Trace("with link results: " & ret)
```

1.6.8 Safe Arrays

A safe array indexes a list of objects. Safe arrays are native to VBScript and provide a way to enumerate all elements in the array. Safe arrays are not native to JScript; they must be converted using the native VBArray function provided by WScript.

Functions that return a safe array value end in Array (for example, EffectivePolicyArray). The followng VBScript and JScript examples use EffectivePolicyArray as a safe array.

JScript Example

```
Action.Trace(" ******** Array Access ****** ");
var a = new VBArray(Query.EffectivePolicyArray());
ret = a.toArray();
for (var i = 0; i < ret.length; i++) {
    var pol = ret[i];
    Action.Trace(" ******** Policy Information ********* ");
    Action.Trace("ID: " + pol.Id);
    Action.Trace("Version: " + pol.Version);
    Action.Trace("Name: " + pol.Name);
    Action.Trace("Type: " + pol.PolicyType);
    Action.Trace("Session: " + pol.Session);
}
```

VBScript Example

```
Dim obj, idx, max, pol
obj = Query.EffectivePolicyArray
Action.Trace VarType(obj)
Action.Trace IsArray(obj)
For Each pol in obj
    Action.Trace " ******** Policy Information ********* "
    Action.Trace "ID: " & pol.Id
    Action.Trace "Version: " & pol.Version
    Action.Trace "Name: " & pol.Name
    Action.Trace "Type: " & pol.PolicyType
    Action.Trace "Session: " & pol.Session
Next
```

1.6.9 Object Match Lists

Because JScript does not support the native importing of safe arrays, and does not support an array enumerator, ZENworks Endpoint Security Management provides an object called Object Match List to allow for index enumeration of a list to both VBScript and JScript. Functions that return this type of object end in List (for example, EffectivePolicyList). The object provides the following functions and properties for access to the objects in the container.

int Count

Description: Returns the number of objects in the container.

object Item(int *idx*)

Description: Returns a particular object from the container based on the index given. If the index is outside the count of container, a null/empty object is returned. The order of objects in the container is not guaranteed.

object Find(string *value*)

Description: Returns an object that matches the value provided. If no matches are found in the container, a null/empty object is returned.

JScript Example

```
Action.Trace(" ******** List Access ****** ");
var ret = Query.EffectivePolicyList;
for(var i = 0; i < ret.Count; i++)
{
    var pol = ret.Item(i);
    Action.Trace(" ******** Policy Information ********* ");
    Action.Trace("ID: " + pol.Id);
    Action.Trace("Version: " + pol.Version);
    Action.Trace("Name: " + pol.Name);
    Action.Trace("Type: " + pol.PolicyType);
    Action.Trace("Session: " + pol.Session);
}
```

VBScript Example

```
set obj = Query.EffectivePolicyList
max = obj.Count
For idx = 0 to (max - 1)
    Action.Trace " ******** Policy Information ********* "
    set pol = obj.Item(idx)
    Action.Trace "ID: " & pol.Id
    Action.Trace "Version: " & pol.Version
    Action.Trace "Name: " & pol.Name
    Action.Trace "Type: " & pol.PolicyType
    Action.Trace "Session: " & pol.Session
Next
```

1.7 Effective Policy Interface

The Endpoint Security Agent evaluates many policies and types to determine which ones will be enforce by a device. Policies that are currently being enforced make up the Effective Policy List.

- Section 1.7.1, “PolicyInformation Object,” on page 28
- Section 1.7.2, “Effective Policies Methods,” on page 29

1.7.1 PolicyInformation Object

The PolicyInformation object provides information about an individual policy in the system. It can be returned by the EffectivePolicyList and EffectivePolicyArray functions.

Data Types: string *Id* — A unique identifier for the policy in the system.

string *Version* — The version of the policy being used.

string *Name* — The name of the policy.

string *PolicyType* — One of the following policy types. Available policy types vary depending on the Endpoint Security Agent version.

- `script`
- `applicationControl`
- `hardware`
- `firewall`
- `locationAssignment`
- `locationRelation`
- `networkEnvironment`
- `security`
- `storageEncryption`
- `storageDeviceControl`
- `usb`
- `vpn`
- `wifi`
- `fde`

string *Session* — The session (`user`, `device`, `zone`) that provided the policy.

Functions: `bool Match(string value)`

Returns `true` if the value provided matches the ID or Name value for the policy.

1.7.2 Effective Policies Methods

The Effective Policies methods get information about a device's currently effective policies.

SafeArray Query.EffectivePolicyArray()

Description: Returns an array of PolicyInformation objects, one for each effective policy being enforced. The list can be empty when there are no published policies. See the example in Section 1.6.8, "Safe Arrays," on page 26.

ObjectMatchList Query.EffectivePolicyList

Description: Returns an array of PolicyInformation objects, one for each effective policy being enforced. The list can be empty when there are no published policies. See the example in Section 1.6.9, "Object Match Lists," on page 27.

JScript Example

```
Action.Trace(" ******** List Access ****** ");
var ret = Query.EffectivePolicyList;
for(var i = 0; i < ret.Count; i++)
{
    var pol = ret.Item(i);
    Action.Trace(" ******** Policy Information ********* ");
    Action.Trace("ID: " + pol.Id);
    Action.Trace("Version: " + pol.Version);
    Action.Trace("Name: " + pol.Name);
    Action.Trace("Type: " + pol.PolicyType);
    Action.Trace("Session: " + pol.Session);
}
```

VBScript Example

```
set obj = Query.EffectivePolicyList
max = obj.Count
For idx = 0 to (max - 1)
    Action.Trace " ******** Policy Information ********* "
    set pol = obj.Item(idx)
    Action.Trace "ID: " & pol.Id
    Action.Trace "Version: " + pol.Version
    Action.Trace "Name: " + pol.Name
    Action.Trace "Type: " + pol.PolicyType
    Action.Trace "Session: " + pol.Session
Next
```

1.8 Location Interface

The Location interface provides methods for getting information about a device's location and for manipulating the location.

- Section 1.8.1, "Definitions," on page 30
- Section 1.8.2, "Data Types," on page 31
- Section 1.8.3, "Security Location Methods," on page 32
- Section 1.8.4, "Mobile (Unknown) Location Methods," on page 35
- Section 1.8.5, "Assigned Location Methods," on page 35
- Section 1.8.6, "Network Location Methods," on page 36
- Section 1.8.7, "JScript Example," on page 36
- Section 1.8.8, "VBScript Example," on page 37

1.8.1 Definitions

ZENworks Endpoint Security Management provides two different lists of locations: a Network Location List and an Assigned Location List. Using these two lists, information about four types of locations is tracked: a Network location, an Assigned location, a Mobile location, and a Security location. A brief description is provided for both of the lists and each location:

- **Network Location List:** Contains all locations defined in the ZENworks Management Zone. These locations may be associated with a set of network environments. The list always contains at least one location that is marked as the Mobile (Unknown) location that is used when the current environment does not match any defined network environments.

- **Assigned Location List:** Contains only the locations that the device is allowed to apply as Security locations. Normally, this list is provided via the Location Assignment policy. This list always contains at least one location that is marked as the Mobile (Unknown) location. The Mobile location is used when the current environment does not match any locations included in the Assigned Location List.
- **Network Location:** The location, taken from the Network Location List, that the current network environment best matches.
- **Assigned Location:** The location, taken from the Assigned Location List, that the current network environment best matches.
- **Security Location:** The location, from the Assigned Location List, that determine which of the security policies are being enforced. Normally, this is the same as the Assigned location. However, scripting or other rules (such as the VPN policy) can force the Security location to change.
- **Mobile Location:** The location, from the Assigned Location List, that has been designated as the default Assigned location if the current network environment does not match any location definitions. This is frequently referred to as the Unknown location.

1.8.2 Data Types

LocationAssignment

The LocationAssignment object provides information about the current location. It is returned when working with a location from the Assigned Location List.

Data Types:

string *Id* — A unique identifier for the location in the system.

string *Name* — The name of the location.

DateTime *DateModified* — The last time the location definition was modified.

int *Order* — The order of precedence between two locations being compared for network environment match.

bool *Mobile* — True if the location is the Unknown location.

bool *AllowsManualChange* — True if the user is allowed to change into or out of this location.

bool *ShowInMenu* — True if the user should see this location listed in the choice of locations menus.

Functions:

`bool Match(string value)`

Returns `true` if the value provided matches the ID or Name value for the location.

LocationNetwork

The LocationNetwork object provides information about the current location. It is returned when working with a location from the Network Location List.

Data Types: string *Id* — A unique identifier for the location in the system.

string *Name* — The name of the location.

DateTime *DateModified* — The last time the location definition was modified.

int *Order* — The order of precedence between two locations being compared for network environment match.

bool *Mobile* — `True` if the location is the Unknown location.

Functions: `bool Match(string value)`

Returns `true` if the value provided matches the ID or Name value for the location.

LocationChange

The LocationChange object provides information about the last location change and why the current location change is being enforced. It is returned when changing the current Security location or can be asked for directly.

Data Types: string *Reason* — One of the following:

- `none` — No change has occurred yet.
- `policy` — A policy update caused the location change.
- `manual` — The location change was manually initiated (for example, by the user).
- `network` — A network environment change caused a match with the new location.
- `rule` — A rule, such as a VPN rule or a script, requested the location change.
- `permanent` — A rule requested a permanent location change. The location change remains in effect until another permanent change is requested or the current request is cancelled.

string *Producer* — The Endpoint Security Agent component that requested the location change. This value can be empty.

string *RuleId* — The ID of the rule that made the location change request. This value an be empty.

string *RuleName* — The name of the rule that made the location change request. This value an be empty.

int *Level* — The level that the request was made.

LocationAssignment *SecurityLocation* — Information about the current Security location resulting from the location change.

1.8.3 Security Location Methods

The Security Location methods deal with the security location, retrieving the current security location, and setting a new location from the script. The Manual location change methods perform the same functions as if the user initiated a request for the location change and follow the same restriction as those put on the user. When the current security location does not allow manual

changes, the script or the user is not able to switch into or out of the location. If the destination location does not allow manual changes, the request is ignored because the location change cannot be switched into by a manual change.

The Rule location change methods allow the script to change from any location to another without restrictions. When a user initiates a manual change, it fails if a location is involved that does not allow manual changes. However, when a script uses the Rule location change (or an internal VPN/Network Environment rule), the location change is allowed regardless of the manual change settings.

The Permanent location change methods allow the script to block changes by internal rules (VPN/Network Environments) and other scripts running in the system. This is done by disabling the location decider code in the Endpoint Security Agent and requiring other scripts/rules to provide the equivalent or higher level before the location can be changed. The internal "VPN" rule in the system uses this method to control location changes when the internet is present. The level it sets is 100.

The final component is the ability to re-enable the location decider. This is controlled by the level setting of the request.

LocationAssignment Query.SecurityLocation

Description: Gets the current Security location.

LocationChange Action.ManualLocationChange(string *toLocation*)

Description: Switches to the *toLocation* if a permanent location has not been set and policy permits.

Parameters: *toLocation* — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy or if policy does not allow manual location changes.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.ManualLocationChangeWithSource(string *fromLocation*, string *toLocation*)

Description: If the current location is the *fromLocation*, switches to the *toLocation* if a permanent location has not been set and policy permits.

Parameters: *fromLocation* — The name or ID of the location being switched from. The request is ignored if the *fromLocation* is not the current location, or if policy does not allow manual location changes.

toLocation — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy or if policy does not allow manual location changes.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.RuleLocationChange(string *toLocation*)

Description: Switches to the *toLocation* if a permanent location has not been set.

Parameters: *toLocation* — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy or if policy does not allow manual location changes.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.RuleLocationChangeWithSource(string *fromLocation*, string *toLocation*)

Description: If the current location is the *fromLocation*, switches to the *toLocation* if a permanent location has not been set.

Parameters: *fromLocation* — The name or ID of the location being switched from. The request is ignored if the *fromLocation* is not the current location, or if policy does not allow manual location changes.

toLocation — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy or if policy does not allow manual location changes.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.PermanentLocationChange(string *toLocation*, int *level*)

Description: Switches to the *toLocation* and turns off the location decider.

Parameters: *toLocation* — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy.

level — The request is permitted only if the current change level is less than or equal to this level.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.PermanentLocationChangeWithSource(string *fromLocation*, string *toLocation*, int *level*)

Description: If the current location is the *fromLocation*, switches to the *toLocation* and turns off the location decider.

Parameters: *fromLocation* — The name or ID of the location being switched from. The request is ignored if the *fromLocation* is not the current location.

toLocation — The name or ID of the location being switched to. The request is ignored if the *toLocation* is not in the Location Assignment policy.

level — The request is permitted only if the current change level is less than or equal to this level.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

LocationChange Action.ReenableLocationDecider(int *level*)

Description: Re-enables the location decider.The location decider waits for a location change event (network environment change, manual change, script, etc.) to occur before making any changes. If you want to change to the current location immediately, you should get the current Assigned location (LocationAssignment Query.AssignedLocation) and assign it as the current Security location (LocationChange Action.PermanentLocationChange(string toLocation, int level) before re-enabling the location decider.

Parameters: *level* — The request is permitted only if the current change level is less than or equal to this level.

Returns: Returns the LocationChange object so the caller can see if the request was honored.

1.8.4 Mobile (Unknown) Location Methods

The Mobile location is often referred to as the Default location or Unknown location. This location is used when no other assigned location matches the current network environment and no rule has overridden the location decider's decisions.

LocationAssignment Query.MobileLocation

Description: Gets the current Mobile location.

Returns: The LocationAssignment object with the current Mobile location information.

1.8.5 Assigned Location Methods

The Endpoint Security Agent is provided a list of locations that it is allowed to use as Security locations. This list is passed to the agent via the Location Assignment policy. The location decider uses this list to determine the best matching location based on the current network environment. That location is called the Assigned location. Scripts cannot change Assigned Locations list or the Assigned location, but they can use it for determining actions and deciding which locations the script may wish to set as the current Security location.

LocationAssignment Query.AssignedLocation

Description: Gets the current Assigned location.

Returns: The LocationAssignment object with the current Assigned location information.

ObjectMatchList Query.AssignedLocationList

Description: Gets the list of Assigned locations available to the device.

Returns: An ObjectMatchList that contains the Assigned locations.

SafeArray Query.AssignedLocationArray()

Description: Gets the list of Assigned locations available to the device.

Returns: A VB SafeArray that contains the Assigned locations.

1.8.6 Network Location Methods

The Endpoint Security Agent receives the list of all locations defined in the ZENworks Management Zone. From this Network Location List, the location decider determines the best location based on the network environment. This is referred to as the Network location. Currently, the ZENworks Adaptive Agent can use this location to determine closest servers and to determine whether or not certain actions (such as bundle downloads) are allowed. A script cannot change the Network location, but it can use the Network location to determine actions, just like the ZENworks Adaptive Agent.

LocationAssignment Query.NetworkLocation

Description: Gets the current Network location.

ObjectMatchList Query.NetworkLocationList

Description: Gets the list of Network locations available to the device.

Returns: An ObjectMatchList that contains the Network locations.

SafeArray Query.NetworkLocationArray()

Description: Returns the list of Network locations available to the device; returned as a Visual Basic SafeArray.

Returns: A VB SafeArray that contains the Network locations.

1.8.7 JScript Example

```
function DisplayAssignedLocation(loc)
{
    Action.Trace("Location = " + loc.Name);
    Action.Trace("Id = " + loc.Id);
    Action.Trace("Date Modified = " + loc.DateModified);
    Action.Trace("Order: " + loc.Order);
    Action.Trace("Mobile: " + loc.Mobile);
    Action.Trace("Allow Manual Change: " + loc.AllowsManualChange);
    Action.Trace("Show in menu: " + loc.ShowInMenu);
}

Action.Trace("");
Action.Trace(" ******** Security Location ********* ");
Action.Trace("");
DisplayAssignedLocation(Query.SecurityLocation);
```

1.8.8 VBScript Example

```
Function DisplayAssignedLocation (loc)
Action.Trace "Location = " & loc.Name
Action.Trace "Id = " & loc.Id
Action.Trace "Date Modified = " & loc.DateModified
Action.Trace "Order: " & loc.Order
Action.Trace "Mobile: " & loc.Mobile
Action.Trace "Allow Manual Change: " & loc.AllowsManualChange
Action.Trace "Show in menu: " & loc.ShowInMenu
End Function

Action.Trace ""
Action.Trace " ******** Security Location ********* "
Action.Trace ""
DisplayAssignedLocation Query.SecurityLocation
```

1.9 Communication Hardware Policy Interface

The Communication Hardware Policy interface provides methods for getting and setting the enforcement for the policy-supported hardware types.

- Section 1.9.1, “Data Types,” on page 37
- Section 1.9.2, “Enforced Policy Methods,” on page 38
- Section 1.9.3, “Hardware Enforcement Methods,” on page 38
- Section 1.9.4, “Adapter Connection Methods,” on page 39
- Section 1.9.5, “JScript Example,” on page 40
- Section 1.9.6, “VBScript Example,” on page 40

1.9.1 Data Types

Hardware Types:	*firewire* — IEEE1394 attached devices
	irda — infrared attached devices
	bluetooth — bluetooth attached devices
	ports — serial or com ports
	modem — modem and dialup adapters
	wireless — wireless network adapters
	wired — wired network adapters
	bridge — network adapter bridges
	any — any of the hardware types

Enforcement Types:

disable — Disable the setting and enforce immediately.

enable — Enable the setting and enforce immediately.

blockConnections — Block connections made by the device; typically applies to wireless network adapters and modems.

blockConnectionsWhenWired — Block connections made by the device only if there is a wired connection.

disableWhenWired — Disable the device when a wired connection is detected.

inherit — Immediately apply enforcement as defined by the current policy/location. Used to clear the script setting.

1.9.2 Enforced Policy Methods

The Enforced Policy methods provide information about whether or not the enforced policy has disabled a specific hardware type.

bool Query.IsHardwareDisabled(string *hardwareType*)

Description: Determines if the enforcement for the specified hardware type is set to `disabled`.

Parameters: *hardwareType* — One of the hardware types listed in Section 1.9.1, “Data Types,” on page 37.

Returns: `True` if the hardware type is disabled by the Endpoint Security Agent. `False` if the agent will allow the hardware type to be enabled and any hardware disabled by the agent should be re-enabled.

1.9.3 Hardware Enforcement Methods

The Hardware Enforcement methods get and set the enforcment for a specific hardware type.

string Query.GetHardwareEnforcement(string *hardwareType*)

Description: Gets the effective enforcement for the specified hardware type.The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Parameters: *hardwareType* — One of the hardware types listed in Section 1.9.1, “Data Types,” on page 37.

Returns: One of the enforcement types listed in Section 1.9.1, “Data Types,” on page 37.

string Query.GetHardwarePolicyEnforcement(string *hardwareType*)

Description: Gets the enforcement, as set by the policy, for the specified hardware type.

Parameters: *hardwareType* — One of the hardware types listed in Section 1.9.1, “Data Types,” on page 37.

Returns: One of the enforcement types listed in Section 1.9.1, “Data Types,” on page 37.

string Query.GetHardwareScriptEnforcement(string *hardwareType*)

Description: Gets the enforcement, as set by script, for the specified hardware type.

Parameters: *hardwareType* — One of the hardware types listed in Section 1.9.1, “Data Types,” on page 37.

Returns: One of the enforcement types listed in Section 1.9.1, “Data Types,” on page 37.

int Action.SetHardwareEnforcement(string *hardwareType*, string *enforcement*)

Description: Sets the enforcement for a specific hardware type.

Parameters: *hardwareType* — One of the hardware types listed in Section 1.9.1, “Data Types,” on page 37.

enforcement — One of the enforcement types listed in Section 1.9.1, “Data Types,” on page 37. These values override the effective policy for the hardware type. If the hardware type does not support the enforcement type (such as `block`, `block_when_wired`, or `disable_when_wired`), enforcement is set to `disable`.

1.9.4 Adapter Connection Methods

The Adaptor Connection methods provide information about whether a specific adapter type has any connections.

bool Query.IsAdapterTypeConnected(string *adapterType*)

Description: Determines if a specific adapter has any connections.

Parameters: *adapterType* — One of the following: `wired`, `wireless`, `modem`, `any`.

Returns: `True` if an adapter of the requested type currently has a connection. `False` if there are no adapters of the requested type with a connection.

1.9.5 JScript Example

```
function DisplayHardwareEnforcement()
{
    Action.Trace("firewire: " + Query.GetHardwareEnforcement("firewire"));
    Action.Trace("wireless: " + Query.GetHardwareEnforcement("bridge"));
}

function SetHardwareEnforcement(enf)
{
    Action.Trace("firewire: " + Action.SetHardwareEnforcement("firewire", enf));
    Action.Trace("wireless: " + Action.SetHardwareEnforcement("wireless", enf));
}

function IsHardwareDisabled()
{
    Action.Trace("firewire: " + Query.IsHardwareDisabled("firewire"));
    Action.Trace("wireless: " + Query.IsHardwareDisabled("wireless"));
}

Action.Trace("");
Action.Trace("Adapter Type Connected:");
Action.Trace("\twireless: " + Query.IsAdapterTypeConnected("wireless"));
Action.Trace("\tany: " + Query.IsAdapterTypeConnected("any"));
Action.Trace("");
Action.Trace("GetHardwareEnforcement:");
DisplayHardwareEnforcement();
Action.Trace("");
Action.Trace("GetHardwarePolicyEnforcement:");
Action.Trace("firewire: " + Query.GetHardwarePolicyEnforcement("firewire"));
Action.Trace("wireless: " + Query.GetHardwarePolicyEnforcement("wireless"));
Action.Trace("");
Action.Trace("GetHardwareScriptEnforcement:");
Action.Trace("firewire: " + Query.GetHardwareScriptEnforcement("firewire"));
Action.Trace("wireless: " + Query.GetHardwareScriptEnforcement("wireless"));
Action.Trace("");
Action.Trace("GetHardwareEnforcement: DisableWhenWired");
DisplayHardwareEnforcement();
Action.Trace("");
Action.Sleep(1000);
 Action.Trace("IsHardwareDisabled: DisableWhenWired");
IsHardwareDisabled();
ret = Action.Prompt("Prompt", "Check for hardware disable when wired", "?", 0, "ok
");
Action.Trace("");
Action.Trace("SetHardwareEnforcement: Inherit");
SetHardwareEnforcement("inherit");
```

1.9.6 VBScript Example

```
Function DisplayHardwareEnforcement()
    Action.Trace("firewire: " & Query.GetHardwareEnforcement("firewire"))
    Action.Trace("wireless: " & Query.GetHardwareEnforcement("wireless"))
 End Function

Function SetHardwareEnforcement(enf)
    Action.Trace("firewire: " & Action.SetHardwareEnforcement("firewire", enf))
    Action.Trace("wireless: " & Action.SetHardwareEnforcement("wireless", enf))
End Function

Function IsHardwareDisabled()
    Action.Trace("firewire: " & Query.IsHardwareDisabled("firewire"))
    Action.Trace("wireless: " & Query.IsHardwareDisabled("wireless"))
End Function

Action.Trace("")
Action.Trace("Adapter Type Connected:")
```

```
Action.Trace("wireless: " & Query.IsAdapterTypeConnected("wireless"))
Action.Trace("any: " & Query.IsAdapterTypeConnected("any"))
Action.Trace("")
Action.Trace("GetHardwareEnforcement:")
DisplayHardwareEnforcement()
Action.Trace("")
Action.Trace("GetHardwarePolicyEnforcement:")
Action.Trace("firewire: " & Query.GetHardwarePolicyEnforcement("firewire"))
Action.Trace("wireless: " & Query.GetHardwarePolicyEnforcement("wireless"))
Action.Trace("")
Action.Trace("GetHardwareScriptEnforcement:")
Action.Trace("firewire: " & Query.GetHardwareScriptEnforcement("firewire"))
Action.Trace("wireless: " & Query.GetHardwareScriptEnforcement("wireless"))
Action.Trace("")
Action.Trace("SetHardwareEnforcement: DisableWhenWired")
SetHardwareEnforcement("disable_when_wired")
Action.Trace("")
Action.Trace("GetHardwareEnforcement: DisableWhenWired")
DisplayHardwareEnforcement()
Action.Trace("")
Action.Sleep(1000)
 Action.Trace("IsHardwareDisabled: DisableWhenWired")
IsHardwareDisabled();
ret = Action.Prompt("Prompt", "Check for hardware disable when wired", "?", 0, "ok
")
Action.Trace("SetHardwareEnforcement: Inherit")
SetHardwareEnforcement("inherit")
```

1.10 WiFi Policy Interface

The WiFi Policy interface provides methods for getting and setting the enforcement for adhoc networks, WiFi connections, and wireless access point security level.

- Section 1.10.1, “Data Types,” on page 41
- Section 1.10.2, “Adhoc WiFi Networks Methods,” on page 42
- Section 1.10.3, “Block WiFi Connections,” on page 43
- Section 1.10.4, “Minimum Security Level Methods,” on page 44
- Section 1.10.5, “Minimum Signal Strength Methods,” on page 44

1.10.1 Data Types

Enforcement Types:	*disable* — Disable the setting and enforce immediately. *enable* — Enable the setting and enforce immediately. *inherit* — Immediately apply enforcement as defined by the current policy. Used to clear the script setting.

Signal Strength: *not_set* — No policy is set; filter is ignored.

very_low

low

good

very_good

excellent

inherit — Immediately apply setting as defined by the current policy. Used to clear the script setting.

Security Level: *inherit* — Immediately apply setting as defined by the current policy. Used to clear the script setting.

unsecured

secure — Any security level.

wep

wpa

wpa2

1.10.2 Adhoc WiFi Networks Methods

The Adhoc WiFi Networks methods get and set the enforcement for adhoc wireless networks.

string Query.GetAdHoc

Description: Gets the effective enforcement for adhoc WiFi networks.The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Returns: `Enabled` if the device can connect to an adhoc wireless network or can be an adhoc network provider. `Disabled` if the device cannot connect to an adhoc network or cannot be a provider.

string Query.GetAdHocPolicy

Description: Gets the enforcement, as set by policy, for adhoc wireless networks.

Returns: `Enabled` if the device can connect to an adhoc wireless network or be an adhoc network provider. `Disabled` if the device cannot connect or be a provider.

string Query.GetAdHocScript

Description: Gets the enforcement, as set by script, for adhoc wireless networks.

Returns: `Enabled` if the device can connect to an adhoc wireless network or be an adhoc network provider. `Disabled` if the device cannot connect or be a provider.

int Action.SetAdHoc(string *enforcement*)

Description: Sets the enforcement for adhoc wireless networks.

Parameters: *enforcement* — One of the enforcement types listed in Section 1.10.1, “Data Types,” on page 41.

1.10.3 Block WiFi Connections

The Block WiFi Connections methods get and set the enforcement for WiFi connections.

string Query.GetBlockWiFiConnection

Description: Gets the effective enforcement for blocking connections to a WiFi network.The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Returns: `Enabled` if WiFi connections are blocked. `Disabled` if WiFi connections are allowed. If disabled, connections are based on availability and filter restrictions.

string Query.GetBlockWiFiConnectionPolicy

Description: Gets the enforcement, as set by policy, for blocking connections to a WiFi network. If disabled, connections are based on availability and filter restrictions.

Returns: `Enabled` if WiFi connections are blocked. `Disabled` if WiFi connections are allowed.If `disabled`, connections are based on availability and filter restrictions.

string Query.GetBlockWiFiConnectionScript

Description: Gets the enforcement, as set by script, for blocking connections to a WiFi network.

Returns: `Enabled` if WiFi connections are blocked. `Disabled` if WiFi connections are allowed. If `disabled`, connections are based on availability and filter restrictions.

int Action.SetBlockWiFiConnection(string *enforcement*)

Description: Sets the enforcement for blocking WiFi connections.

Parameters: *enforcement* — One of the enforcement types listed in Section 1.10.1, “Data Types,” on page 41.

1.10.4 Minimum Security Level Methods

Minimum security level is used to filter out wireless networks that do not meet the minimum level. Devices cannot see or connect to the removed wireless networks. The security level is inclusive from inherit to wpa2, as listed in Section 1.10.1, “Data Types,” on page 41. For example if wpa is chosen, networks that support wpa and wpa2 security pass the filter, but unsecured networks and wep networks are filtered out.

The Minimum Security Level methods get and set the minimum security level requirement for a wireless network.

string Query.GetMinWiFiSecurityLevel

Description: Gets the effective enforcement for the minimum security level. The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Returns: One of the security levels listed in Section 1.10.1, “Data Types,” on page 41.

string Query.GetMinWiFiSecurityLevelPolicy

Description: Gets the minimum security level, as set by policy.

Returns: One of the security levels listed in Section 1.10.1, “Data Types,” on page 41.

string Query.GetMinWiFiSecurityLevelScript

Description: Gets the minimum security level, as set by script.

Returns: One of the security levels listed in Section 1.10.1, “Data Types,” on page 41.

int Action.SetMinWiFiSecurityLevelEnforcement(string *enforcement*)

Description: Sets the enforcement for minimum security level.

Parameters: *enforcement* — One of the enforcement types listed in Section 1.10.1, “Data Types,” on page 41.

1.10.5 Minimum Signal Strength Methods

Minimum signal strength level is used to filter out wireless access points that do not meet the minimum signal strength. Devices cannot see or connect to the removed access point. The signal strength is inclusive from `inherit` to `not_set`, as listed in Section 1.10.1, “Data Types,” on page 41.

For example if `very_good` is chosen, access points that have `very_good` and `excellent` signal strength pass the filter, but access points with `very_low`, `low`, and `good` signal strengths are filtered out.

The Minimum Signal Strength methods get and set the minimum signal strength requirement for wireless access points.

string Query.GetMinWiFiSignalStrength

Description: Gets the effective enforcement for the minimum signal strength. The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Returns: One of the signal strengths listed in Section 1.10.1, “Data Types,” on page 41.

string Query.GetMinWiFiSignalStrengthPolicy

Description: Gets the minimum security level, as set by policy.

Returns: One of the signal strengths listed in Section 1.10.1, “Data Types,” on page 41.

string Query.GetMinWiFiSignalStrengthScript

Description: Gets the minimum security level, as set by script.

Returns: One of the signal strengths listed in Section 1.10.1, “Data Types,” on page 41.

int Action.SetMinWiFiSignalStrengthEnforcement(string *enforcement*)

Description: Sets the enforcement for minimum security level.

Parameters: *enforcement* — One of the enforcement types listed in Section 1.10.1, “Data Types,” on page 41.

1.11 Storage Device Control Policy Interface

The Storage Device Control Policy interface provides methods for getting and setting the enforcement for different volume types (fixed, optical, removable, and floppy), and for getting and setting the enforcement for the AutoPlay and AutoRun features.

- Section 1.11.1, “Data Types,” on page 46
- Section 1.11.2, “AutoPlay Methods,” on page 46
- Section 1.11.3, “Volumes Methods,” on page 47

1.11.1 Data Types

Volume Types:	*unknown* — Volume drive type cannot be determined. *fixed* — Local hard drive located on a removable system bus. *optical* — CD-ROM and DVD drives. *removable* — Volumes on a removable bus or volumes marked as removable by the system. *floppy* — Floppy disk drives.
Volume Access:	*inherit* — Immediately apply setting as defined by the current policy. Used to clear the script setting. *disable* — Block all access to the volume. Disable in Device Manager. *deny* — Block read and write access to the volume, but leave volume enabled in Device Manager. *read_only* — Allow the volume to be read from, but block write operations. *read_write* — Allow full access to the volume.
Auto-Play Access:	*inherit* — Immediately apply setting as defined by the current policy. Used to clear the script setting. *allow* — Allow Windows to initiate an auto-play (or auto-run) request when mounting a volume. *block_auto_play* — Do not allow Windows to initiate an auto-play (or auto-run) request when mounting a volume. *block_auto_run* — Do not allow Windows to initiate an auto-run request when mounting a volume; auto-play requests are allowed.
Enforcement Type:	*disable* — Disable the setting and enforce immediately. *enable* — Enable the setting and enforce immediately. *inherit* — Immediately apply enforcement as defined by the current policy. Used to clear the script setting.

1.11.2 AutoPlay Methods

The AutoPlay methods get and set the enforcement for the AutoPlay and AutoRun features.

string Query.GetAutoPlayEnforcement

Description:	Gets the enforcement for auto-play.
Returns:	One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

string Query.GetAutoPlayPolicyEnforcement

Description: Gets the auto-play enforcement type, as set by policy.

Returns: One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

string Query.GetAutoPlayScriptEnforcement

Description: Gets the auto-play enforcement type, as set by script.

Returns: One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

int Action.SetAutoPlayEnforcement(string *enforcement*)

Description: Sets the enforcement for auto-play.

Parameters: *enforcement* — One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

1.11.3 Volumes Methods

The Volumes methods get and set the enforcement for fixed, optical, removable, and floppy volumes.

string Query.GetVolumeEnforcement(string *volumeType*)

Description: Gets the effective enforcement for volumes of the specified type. The effective enforcement is determined by resolving any conflicts between the policy enforcement type and the script enforcement type. The script enforcement type overrides the policy enforcement type; if the script enforcement type is `inherit`, the policy enforcement type is used.

Parameters: *volumeType* — One of the volume types listed in Section 1.11.1, “Data Types,” on page 46.

Returns: One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

string Query.GetVolumePolicyEnforcement(string *volumeType*)

Description: Gets the enforcement for volumes of the specified type, as set by policy.

Parameters: *volumeType* — One of the volume types listed in Section 1.11.1, “Data Types,” on page 46.

Returns: One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

string Query.GetVolumeSciptEnforcement(string *volumeType*)

Description: Gets the enforcement for volumes of the specified type, as set by script.

Parameters: *volumeType* — One of the volume types listed in Section 1.11.1, “Data Types,” on page 46.

Returns: One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

int Action.SetVolumeEnforcement(string *volumeType*, string *enforcement*)

Description: Sets the enforcement for volumes of the specified type.

Parameters: *volumeType* — One of the volume types listed in Section 1.11.1, “Data Types,” on page 46 except for `fixed`. You cannot set an enforcement type for a fixed volume.

enforcement — One of the enforcement types listed in Section 1.11.1, “Data Types,” on page 46.

2 Script Testing

You can use the Endpoint Security Agent to test scripts. You can test an unpublished script as part of the script development process, or you can test a published Scripting policy in order to troubleshoot problems.

The following sections provide information to help you test scripts. The sections do not include information about creating scripts; for that information, see Appendix 1, "Script Development," on page 7.

- Section 2.1, "Enabling Script Testing in the Endpoint Security Agent," on page 49
- Section 2.2, "Testing an Unpublished Script," on page 49
- Section 2.3, "Testing a Published Scripting Policy," on page 53
- Section 2.4, "Tracing a Script's Execution," on page 55

2.1 Enabling Script Testing in the Endpoint Security Agent

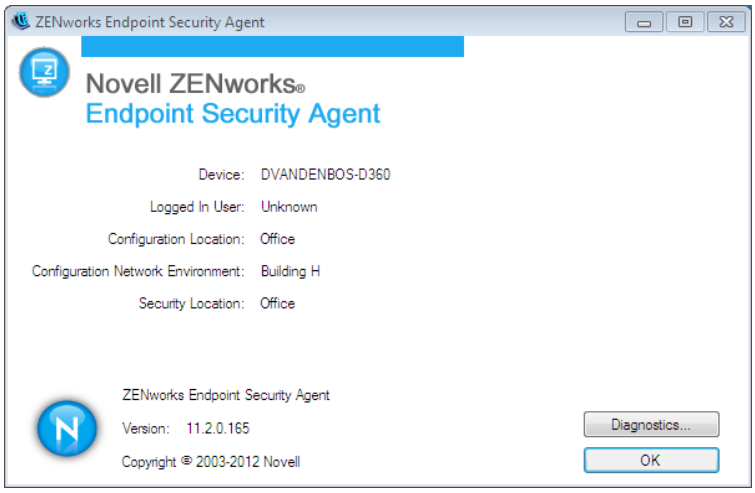
To access the script testing features in the Endpoint Security Agent, you must provide an override password. The override password is configured in ZENworks Control Center as one of the ZENworks Agent configuration settings (ZENworks Control Center > *Configuration* > *Management Zone Settings* > *Device Management* > *ZENworks Agent*). For information about setting the override password, see "ZENworks Agent Settings"in the *ZENworks 11 SP2 Adaptive Agent Reference*.

2.2 Testing an Unpublished Script

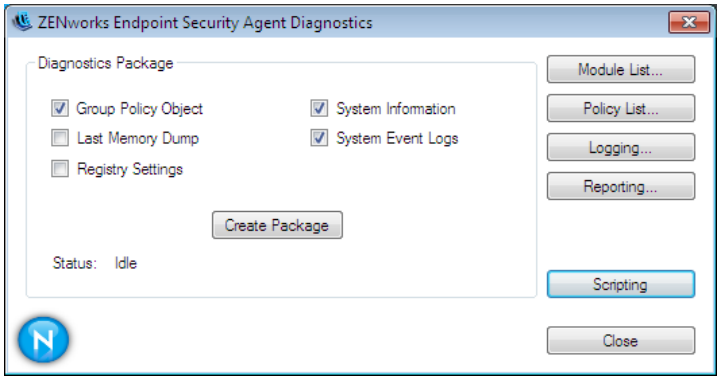
The following steps explain how to test a script that is not yet published in a Scripting policy. If you want to test a script that has already been published to a device as a Scripting policy, see Section 2.3, "Testing a Published Scripting Policy," on page 53.

1. Make sure that the script testing features of the Endpoint Security Agent are enabled for the device where you plan to test the script. For details, see Enabling Script Testing in the Endpoint Security Agent.
2. On the device, double-click the icon in the notification area, then click *Endpoint Security*.

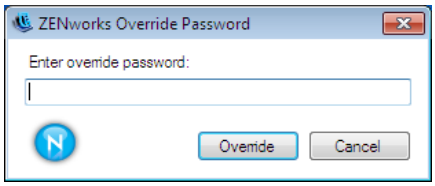
3 In the *Endpoint Security Agent Actions* section, click *About* to display the About dialog box.



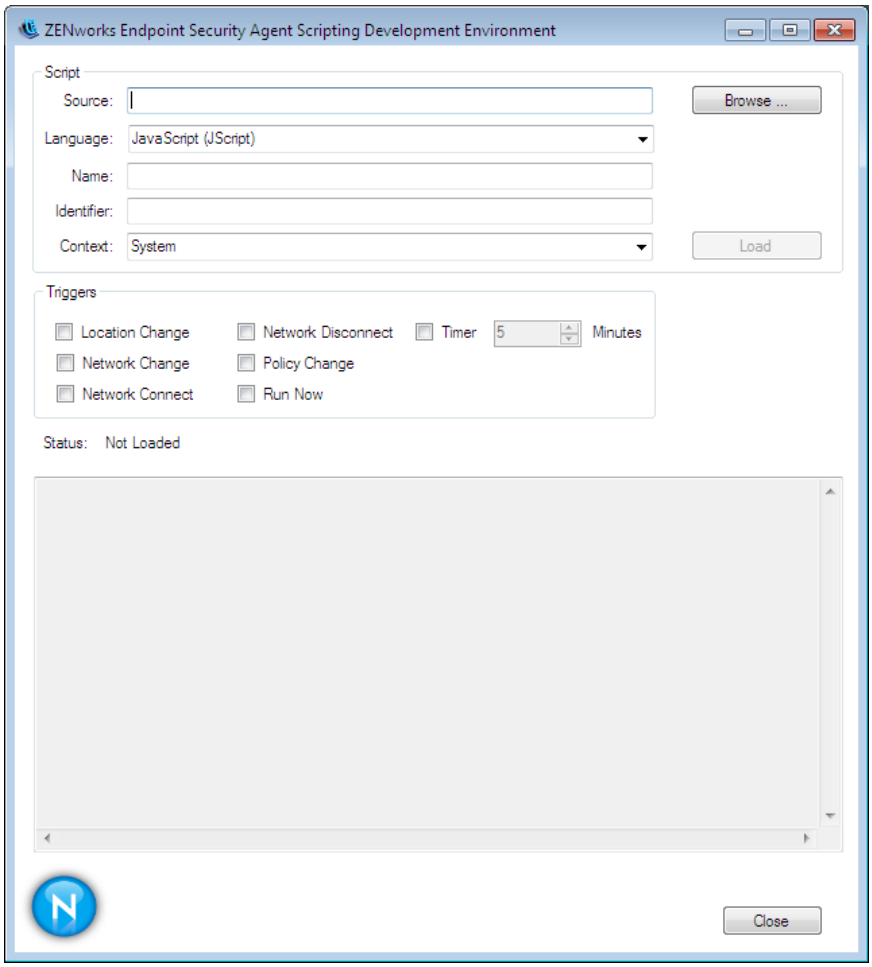
4 Click *Diagnostics*.



5 Click *Scripting* to display the override password prompt.



6 Specify the override password, then click *OK* to display the ZESM Scripting Development Environment dialog box.



7 In the Source field, click Browse, select the script you want to test, then click Open

The script source, language, name, and identifier are displayed.

8 In the Context field, select the context in which you want the script to run.

9 In the Triggers section, select the execution triggers to test.

Location Change: Triggers script when any location change occurs.

Network Change: Triggers script when any network environment change occurs.

Network Connect: Triggers script when any network (wireless, wired, modem/dialup) connection occurs.

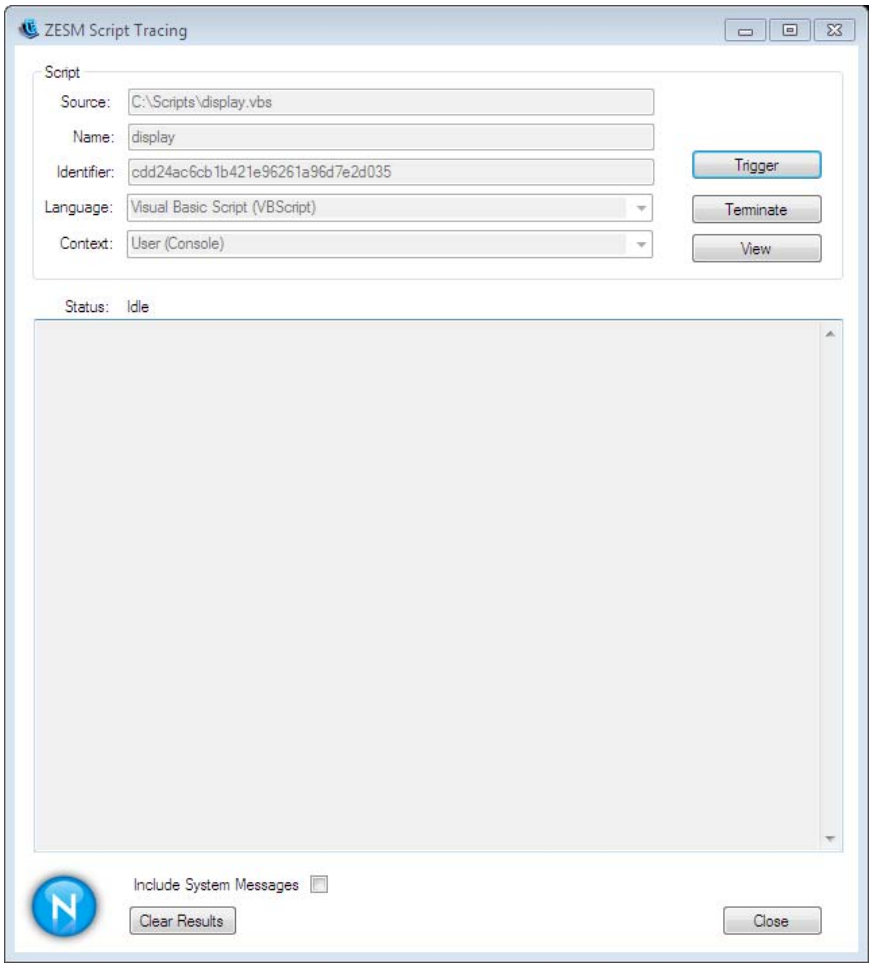
Network Disconnect: Triggers script when any network (wireless, wired, modem/dialup) disconnect occurs.

Policy Change: Triggers script when any Security policy change is received.

Run Now: Triggers script immediately upon loading of the script.

Timer: Triggers script at the specified interval.

2 Click *Trace* to display the ZESM Script Tracing dialog box.



3 Select *Include System Messages* if you want to include output for all actions.

If you do not include system messages, only messages generated by `Action.Trace` commands in the script are output. For more information about using `Action.Trace` commands, see "void Action.Trace(string msg)" on page 14.

4 Click *Trigger* to execute the script.