CHAPTER 8

Using Active Monitors

Active Monitors query network services installed on a device, then wait for the response. If a response is not received or if the response does not match what is expected, the service is considered down, and a state change occurs on the device. If the query is returned with the expected response, the service is considered up. The following active monitor types are available in Cisco netManager:

- DNS Monitor
- NT Service Monitor
- Ping Monitor
- Active Script Monitor
- SNMP Monitor
- TCP/IP Monitor
- Telnet Monitor

Note There are several types of TCP/IP Monitors that are configured using the same dialog box.

About Monitors and Actions

The monitors and the action systems work together in Cisco netManager to help you stay informed about what is happening on your network and the devices connected to your network. It is a cooperative relationship that can be configured to go well beyond the default setting included with the installation of the product. With some helpful examples and some creativity, a network administrator should be able to tailor the monitors and actions systems to watch over all of their important devices and troubleshoot problems that may arise.

Using Monitors and Actions

When you set up actions and monitors, keep in mind the following, to help you maximize the usefulness of the features and minimize problems.

- **Action Coverage**. Set up your notification actions so that the people who have to be notified are sent the alert. Consider creating vacation action types that will not send alerts to people who cannot do anything about it.
- **Understanding SNMP**. It will take a little research, but when you find out which of your network devices have SNMP capabilities, you can configure monitors to listen for all types of information and trigger an Action accordingly.
- **Security Features**. Pay careful attention to the devices and services that are critical to your network security.
- **Network Resources**. Configure Cisco netManager to perform an action when your network resource availability diminishes across a certain threshold.
- **Assigning Actions to Devices or Monitors**. Assign actions to the device if you only want one notification when the device goes down. Assign an action to a specific active monitor if it requires special attention.

About the Active Monitor Library

The Active Monitor Library is the central storehouse of all active monitors that have been configured for your network. When changes are made to the active monitors listed in this dialog box, the changes affect each instance of that particular monitor across your device groups.

Access the Active Monitor Library from the main menu of the Cisco netManager. In the web interface, click **GO** > **Configure** > **Active Monitor Library**.

This dialog box is used to configure new or existing active monitor types. The list shows all types currently configured for use in Cisco netManager. From here, you can do the following:

- To configure a new active monitor type, click **New**.
- To change the current configuration of an active monitor type, select an existing type, then click **Edit**.
- To make a copy of that type and add it to the list, select an active monitor type, then click **Copy.**
- To remove an active monitor type, select it from the list, then click **Delete**.
- In the Cisco netManager console, you can select an active monitor, then click **Test** to test the selected active monitor on a device.

Supported Active Monitor Types

The following is a list of all of the active monitor types that are supported by Cisco netManager:

- **Active Script Monitor**—The Active Script Monitors let you write either VBScript or JScript code to perform a check on a device. If the script returns an error code, the monitor is considered down.

Note Cisco netManager does not support the scripts that you create, only the ability to use them in the Active Script Monitor.

- **DNS Monitor**—The DNS monitor checks for the Domain Name Server (DNS) on port 53. If no DNS service responds on this port, then the service is considered down.

- **SNMP Monitor**—Simple Network Management Protocol is the protocol governing network management and monitoring of network devices and their functions. This monitor queries the SNMP device and tries to match the expected returned value.
- **Telnet Monitor**—Telnet is a simple service monitor that checks for a Telnet server on port 23. If no Telnet service responds on this port, then the service is considered down.
- **Ping Monitor**—The Ping Monitor sends an ICMP (ping) command to the device. If the device does not respond, the monitor is considered down.
- **TCP/IP Monitor**—The TCP/IP Monitor is used to monitor a TCP/IP service that either does not appear in the list of standard services or uses a nonstandard port number.
- **NT Service Monitor**—The NT Service Monitor lets you check the status of a service on a Windows machine and attempts a restart of the service (if the appropriate Administrator permissions exist).

Note A running Windows Management Instrumentation (WMI) service on the targeted machine is required for this NT Service Monitor to work properly. Windows 2000 Service Pack 2 or higher, XP, and 2003 are installed with the WMI service. WMI is not installed with Cisco netManager, but can be downloaded from Microsoft and installed on Windows NT.

Assigning Active Monitors

There are two steps in assigning an active monitor to a device. The first is to configure the active monitor in the Active Monitor Library, and the second is to add that monitor to a device. For most users, the default configuration is sufficient and there is no need to make any changes to the active monitors in the library.

To configure an active monitor:

Step 1 From the web interface main menu, select **GO > Configure > Active Monitor Library** to view the Active Monitor Library.

Step 2 Do one of the following:

- Click **New** to configure a new active monitor.
- Select a monitor from the list and click **Edit** to make changes to an existing configuration.

Step 3 After you make the necessary changes, click **OK** to add the monitor to the list, or to save the changes you made to one already on the list.

To add an active monitor to a device:

Step 1 Select the active monitors you want to scan for during Device Discovery. When you select the discovered devices and add them to your database, Cisco netManager creates a monitor for each network service found.

Step 2 In the Device Properties Active Monitor dialog box, click **Discover**. Cisco netManager scans the device and creates a monitor for each network service found.

Step 3 Manually assign an active monitor to the device:

a. In the Device Properties Active Monitor dialog box, click **Add**. The Active Monitor Properties dialog box opens.

b. Select the active monitor type you want to assign to the device, then click **Next**.

c. Set the polling properties for the monitor, then click **Next**.

d. Setup actions for the monitor state changes.

e. Click **Finish** to add the monitor to the device.

Step 4 Add a new device. Click **GO** > **Devices** > **New Device**. The Add New Device dialog box opens.

Step 5 Click **Advance**. The Device Discovery Properties dialog box opens.

Step 6 In the **Select Active Monitors to be used in the scan process** section, select the active monitor types you want to assign to the device.

Step 7 Click **OK**.

Step 8 Use **Bulk Field Change** to add an active monitor to multiple devices:

a. Select the devices in the device list, then right-clicke of the selected items.

b. From the right-mouse menu, select **Bulk Field Change** > **Active Monitor**.

c. Select the active monitor type you want to add.

d. Click **OK**.

Deleting Active Monitors

Unless you are absolutely sure you need to remove an active monitor type from the Active Monitor Library, you should never have to delete an item from this list. If you do, and you find you need it later, you will have to reconfigure it completely, including the default types that were added during initial installation of Cisco netManager. We recommended that you only delete the custom monitors that you create.

Caution When you remove an active monitor type from the library, all active monitors of that type are deleted from the devices you are monitoring, and all related report data is lost.

The best course of action is to remove the monitors at the device level or to disable the monitor by clearing the selection on the Device Properties.

To remove a monitor from a device:

Step 1 Right-click the device you want to remove the monitor from, then click **Properties**. The Device Properties dialog box opens.

Step 2 Click **Active Monitors**. The active monitors attached to the selected device displays in the list.

Step 3 Select the monitor you want to remove.

Step 4 Click **Remove**. A warning dialog box opens, stating that all data for that monitor will be deleted if the monitor is removed.

Step 5 Click **Yes** to remove the monitor.

Note If you want to stop monitoring an active monitor on a device, but want to keep the historical data, then you must disable the monitor instead of deleting it from a device.

Using the Bulk Field Change Feature

To remove an active monitor from multiple devices:

Step 1 Select the devices in the Device View or Map View, then right-click one of the selected items. The context menu opens.

Step 2 Select **Bulk Field Change > Active Monitor**. The Bulk Field Change: Active Monitor dialog box opens.

Step 3 In the **Operation** list, click **Remove**.

Step 4 In the **Active Monitor type** list, select the active monitor that you want to remove.

Step 5 Click **OK** to remove the monitor from the selected devices.

Group and Device Active Monitor Reports

The following reports display information for devices or device groups that have active monitors configured and enabled. Access these reports from the Reports tab on the web interface.

- State Change Acknowledgement
- Active Monitor Availability
- Active Monitor Outage
- Health
- State Change Timeline
- State Summary
- Device Status

Example: Monitoring Network Printer Toner Levels

To avoid running out of printer ink in the middle of print jobs, or wasting toner by switching toner cartridges before they are empty, through Cisco netManager you can create a custom SNMP active monitor that will notify you when toner levels are low.

To configure the printer monitor:

Step 1 From the Cisco netManager web interface, click **GO > Configure > Active Monitor Library**. The Active Monitor Library dialog box opens.

You need to create an active monitor for each printer type in use. It may be that the office uses the same printer type in each office. In this example, we are using a Hewlett Packard LaserJet 4050N. Check your network printers for their specific maximum capacity toner levels.

Step 2 Click **New**, select **SNMP Monitor**, then click **OK**. The Add SNMP Monitor dialog box opens.

Step 3 Enter a Name and Description for the monitor; for example, TonerMonitor and Toner monitor for the Hewlett Packard LaserJet 4050N.

For the **Object ID** and **Instance**, click the browse (**...**) button; then locate and find the **prtMarkerSuppliesLevel (OID 1.3.6.1.2.1.43.11.1.1.9) SNMP** object in the MIB object tree. This SNMP object is found in the MIB tree at:

- mgmt
  - mib 2
    - printmib
      - prtMarkerSupplies
      - prtMarkerSuppliesEntry
      - prtMarkerSuppliesLevel

Step 4 Select **Range of Values** from the drop-down menu and enter 4600 (the maximum capacity toner level) as the high value and 100 as the low value, then click **OK**. The action will fail when the printer toner level reaches 99.

Step 5 Test the newly created active monitor and make appropriate changes, if needed.

Step 6 Assign the active monitor to the printer device by clicking **Device Properties > Active Monitors**.

Step 7 In the Active Monitor dialog box, click **Add**.

Step 8 During the configuration wizard, create or select an action to notify you when the printer's toner levels are low.

Step 9 Repeat steps 6 through 8 for each network printer that requires monitoring.

Expression Editor

Cisco netManager knows the proper connecting commands for checking the standard services listed on the Services dialog box, but to monitor a custom service, you may want to specify what commands to send to the service and what responses to expect from the service in order for Cisco netManager to consider the service up. It is up to you to determine the proper command strings to expect and send for a custom service.

You can use a rule expression to test a string of text for particular patterns.

Script Syntax

You create a script using keywords. In general, the Script Syntax is **Command=String**. Command is either **Send**, **Expect**, **SimpleExpect** or **Flow Control**.

Note A script can have as many send and receive lines as needed. However, the more you have, the slower the service checking.

Keywords

- To send a string to a port, use the Send= keyword.
- To expect a string from a port, use the SimpleExpect= or the Expect= keyword.
- To comment out a line, use the # symbol as the first character of the line.
- To have conditional responses on "error" or "success" of a step within the scripts, use Flow Control Keywords.

Examples

You have a TCP service to check, where you need to do the following:

- Expect something on connection
- Send a command
- Check for a response
- Send something to disconnect

Script Syntax: Expect=Keyword

This keyword provides you a great amount of flexibility to accept variable responses and choose only the information you need. This is accomplished using special control characters and regular expressions. If you do not need all this flexibility or are new to writing your own custom TCP/UDP scripts, then you may want to start off using the SimpleExpect keyword first.

Variations of the Expect Keyword

There are four variations of the Expect keyword:

- **Expect**. Returns true when the expected value is matched.
- **Expect(MatchCase)**. Returns true only when the case matches the expected value.
- **DontExpect**. Returns true when the value is not found.
- **DontExpect(MatchCase)**. Returns true when the value is not found.

The Expect syntax has the form `Expect=Response` where the Response is specified either as an exact text string or as a mixture of regular expression rules and text. The Add/Edit Expect Rule button will help you construct and test a regular expression response string. It will automatically choose the variation of Expect for you based on options you select in that dialog box. The Add/Edit Expect Rule button does not aid in the generation of SimpleExpect keywords.

Note The `~`, `^`, `!` and `= =` codes have been replaced with variations on the Expect keyword itself. Migrated definitions will be converted automatically.

```
Example 1:
#
# Note: script comments start with a # character
```

```
#
# Send a simple text command
#
Send = Hello There
#
# Expect a nice response that begins with, "Hi, How are you"
#
Expect=^Hi, How are you

Example 2:
#
# Send a command followed by CR/LF
#
Send=Select * from Accounts\r\n
#
# Expect a large response, but we only care to check that somewhere
# in the response John Doe is mentioned
#
Expect=John Doe

Example 3:
#
# Send a binary escape (27) and an x y and z and then a nak (21)
#
Send=\x1Bxyz\x15
#
# Expect something that does *not* contain 123 escape (27)
#
DontExpect=123\x1B
```

Script Syntax: Flow Control Keywords

The script language has been expanded to have conditional responses on "error" or "success" of a step within the scripts. This is done by using the following keywords:

- **IfState**. Checks for the current state (ok or error) and jumps to a label if true.
  Valid syntax: `IfState {ERR|OK} label`
  Example:
  `IfState ERR End`
  `IfState OK Bye`
- **Goto**. This immediately jumps to a label.
  Valid syntax: `Goto End`
  Example:
  `Goto End`
- **Exit**. This immediately ends the script with an optional state (ok or error). The optional state overrides the current state.
  Valid syntax: `Exit {ERR|OK}`
  Example:
  `Exit ERR`
  `Exit OK`
- **:Label**. This defines a label that can be the target of a jump. A label is defined by a single word beginning with the ":" character.
  Valid syntax: `:`(with a name following)
  Example:
  `:Bye`

- **OnError**. This allows for a global handling of an error situation.
  Valid Syntax: `OnError {EXIT|CONTINUE|GOTO} label`

  Example:

  ```
  OnError EXIT (Default behavior)
  OnError CONTINUE
  OnError GOTO Logoff
  ```

Script Syntax: Send=Keyword

To send a command on a connection, use a Send=keyword. The form is Send=Command. The command is exactly the message you want to send. You may use a combination of literal characters and binary representations.

Cisco netManager understands the C0 set of ANSI 7-bit control characters. A binary can be represented as `\x##`, where ## is a hexadecimal value. Those familiar with the table may also choose to use shorthand such as `\A (\x01)` or `\W (\x17).`

You can use `\r` and `\n` as the conventions for sending the carriage return and line feed control characters to terminate a line.

The following table shows the keywords you can use.

Keyword	Description
\x##	Binary value in hexadecimal; for example, \x1B is escape
\\	The "\" character
\t	The tab character (\x09)
\r	The return character (\x0D)
\n	The new line character \x0A)

Note The %### decimal syntax for specifying binary octets has been replaced with the \x## hexadecimal syntax. Migrated definitions will be converted automatically.

```
Example 1:
#
# Note: script comments start with a # character
#
# Send a simple text command
#
Send=Hello There

Example 2:
#
# Send a command followed by CR/LF
#
Send=Select * from Accounts\r\n
```

```
Example 3:
#
# Send a binary escape (27) an x y and z and then a nak (21)
#
Send=\x1Bxyz\x15
```

Script Syntax: SimpleExpect Keyword

The SimpleExpect keyword lets you specify expected responses from your server. Responses can be binary (that is, nonprintable ASCII character) responses. If you know exactly (or even approximately) what to expect you can construct a simple expect response string to match against.

This keyword allows you some flexibility in accepting variable responses and choosing only the information you need. If you need additional flexibility you may want to consider using the regular expression syntax available in the Expect Keyword.

The SimpleExpect form is SimpleExpect=Response. Where the response is just a series of characters you expect back from the service. The following table displays keywords that match logic and wildcards to compare responses byte by byte expanding escape codes as you go.

Command Options

Keyword	Description
\x##	Binary value in hexadecimal; for example,\x00 is null
.	Matches any character
\%	The "%" character
\.	The "**.**" character
\\	The "\" character

Note Only the number of characters specified in the expect string are used to match the response. The response is expected to start with these characters. Any extra trailing characters received are just ignored.

```
Example 1:
#
# Note: script comments start with a # character
#
# Send=Hello There
#
# Expect a nice response
#
SimpleExpect=Hi, how are you?

Example 2:
#
# Send a command followed by CR/LF
#
Send=Select * from Accounts\r\n
#
# Expect a large response, be we only care to check that first word
# received is "Customer"
#
SimpleExpect=Customer

Example 3:
#
# Send a binary escape (27) an x y and z and then a nak (21)
#
Send=\x1B\x15
#
# Expect any byte (we don't care) then an abc and an ack (6)
#
SimpleExpect=.abc\x06
```

Send to Disconnect Examples

For a service such as FTP, the command would be **`QUIT/r/n`**. If a command string is not specified, the connection is closed by sending a FIN packet and then an RST packet.

The /r (carriage return) and /n (line feed) are the conventions for sending these control characters to terminate a string. You can use:

- /r = 0x0a
- /n = 0x0d
- /t = 0x09 or /xnn where nn is any hexadecimal value from 00 to FF

The disconnect string is:

Send=QUIT/r/n

Regular Expression Syntax

The following tables list the syntax understood by the Cisco netManager Regex Engine:

Table 8-1 Matching a Single Character

Meta-Character	Description	Matches
`.`	Dot	Matches any one character.
`[...]`	Character class	Matches any character inside the brackets. For example, [abc] matches a, b, and c.
`[^...]`	Negated character class	Matches any character except those inside the brackets. For example, [^abc] matches all characters except a, b, and c.
`-`	Dash	Used within a character class. Indicates a range of characters. For example: • [2-7] matches any of the digits 2 through 7. • [0-3a-d] is equivalent to [0123abcd].
`\`	Escaped character	Interpret the next character literally. For example, 3\.14 matches only 3.14, whereas 3.14 matches 3234, and so on

\xnn	Binary character	Match a single binary character. nn is a hexadecimal value between 00 and FF. For example: • \x41 matches A. • \x0B matches Vertical Tab.

Table 8-2 Quantifiers

Meta-Character	Description	Matches
?	Question	One optional. The preceding expression once or not at all. For example, colou?r matches colour or color. For example, [0-3][0-5]? matches 2 and 25.
*	Asterisk	Optional. Any number allowed. For example, .* Zero or more occurrences of any character.
+	Plus	One required, additional are optional. For example, [0-9]+ matches 1, 15, 220, and so on.
??, +?, *?		Nongreedy versions of ?, +, and *. Match as little as possible, whereas the greedy versions match as much as possible. For example, for input string <html>content</html>. <.*?> matches <html> <.*> matches <html>content</html>

Table 8-3 Matching Position

Meta-Character	Description	Matches
^	Caret	Matches the position at the start of the input. For example: • ^2 will only match input that begins with 2. • ^[45] will only match input that begins with 4 or 5.
$	Dollar	At the end of a regular expression, this character matches the end of the input. For example, >$ matches a right arrow symbol (>) at the end of the input.

Table 8-4 Other

Meta-Character	Description	Matches
|	Alternation	Matches either of the expressions that it separates. For example, H|Cat matches either Hat or Cat.
(...)	Parentheses	Provides grouping for quantifiers; limits scope of alternation via precedence. For example, (abc)* matches 0 or more occurrences of the the string abc.

\0, \1, ...	Backslash	Matches text previously matched within first, second (and so on) match group (starting at 0). For example, <{head}>.*?</\0> matches "<head>xxx</head>".
!	Negation	The expression following ! does not match the input. For example, a!b matches a not followed by b.

Table 8-5 Abbreviations

Abbreviation	Matches
\a	Any alphanumeric character: (A-Z, a-z, 0-9).
\b	White space (blank): ([\\t]).
\c	Any alphabetic character: ([a-zA-Z]).
\d	Any decimal digit: [0-9].
\D	Any nondecimal digit [^0-9].
\h	Any hexadecimal digit: ([0-9a-fA-F]).
\n	New line: (\r|(\r?\n)).
\p	Any punctuation character: ,./\';:"!?@#$%^&*()[]{}- _=+|<>!~.
\P	Any nonpunctuation character.
\q	A quoted string: (\"[^\"]*\")|(\'[^\']*\').
\s	Cisco netManager-style white space character [\\t\\n\\r\\f\\v].
\S	Cisco netManager-style nonwhite space character [^ \\t\\n\\r\\f\\v].
\w	Part-of-word character ([a-zA-Z0-9_]).
\W	nonword character ([^a-zA-Z0-9_]).
\z	An integer: ([0-9]+).

Text String Example

To check an Internet Relay Chat (IRC) service, you can send the command **Version/r/n;** and the expected response from the IRC service is irc.

```
Name: IRC; Port: 6667; TCP.
Send=Version/r/n
Expect=irc
Send=QUIT/r/n
```

Note You can use Telnet to find the proper value for **SimpleExpect**, or an **Expect** string for a particular service. Packet capture tools can also be very useful.

Using Telnet to Determine "Expect on Connect" String

Telnet to the desired port on the host when you are certain it is working properly, and see what comes back. You can enter just an identifying portion of a SimpleExpect or Expect keyword.

For example, if you expect to get "220 hostname.domain.com Imail v1.3" back from the host, you could use "220 host" as a response string (that is, `SimpleExpect=220 host`, or `Expect=^220 host`).

Note Some services are based on binary protocols (such as DNS) and will not provide you with a simple response string to use. You can use a packet capture tool to view these types of responses.

Using Active Script Monitor

The Active Script Monitors let you write either VBScript or JScript code to perform a check on a device. If the script returns an error code, the monitor is considered down.

Note Cisco netManager does not support the scripts that you create, only the ability to use them in the Active Script Monitor.

- **Name**. The name of the monitor as it appears in the Active Monitor Library.
- **Description**. The description of the monitor as it appears in the Active Monitor Library.
- **Timeout**. The amount of time (in seconds) Cisco netManager should wait for a response to the poll.

Note Though the maximum timeout is 60 seconds, you should not use a timeout longer than the default of 10 seconds. You should use the shortest timeout possible.

- **Script type**. VBScript or JScript
- **Script text**. Write or insert your monitor code here.
- **Use in discovery**. Select this option to have the monitor appear in the active monitor list during discovery. From there, you can select the monitor to have Cisco netManager discover that monitor type in your devices.

This script monitor has a context object that you can use to poll for specific information about the device.

Examples: Active Script Monitor Context Code

The following table lists several examples of active script monitor context code that you can use to create useful active monitors for your devices. To use these examples, select the text of the context and then copy and paste the code into the **Script text** box of the Active Script Monitor dialog box.

Note If the copyright information appears in the text that you copied and pasted from the filter, you should delete it.

Table 8-6 Context Code Examples

Monitor	Code

How to return the results of the script to Cisco netManager.
Note This affects the state of the device.
	JScript:
	`Context.SetResult(0, "    Everything is OK"); //Success`
	`Context.SetResult(1, "    Really big big error"); //Failure`
	VBScript:
	`Context.SetResult 1, "    Really big big error"`

Logging a message to the Cisco netManager event viewer. **Note** To view Context.LogMessage entries, you must have selected **Debug On** in the event viewer.	JScript: `Context.LogMessage("This is the message");`
Accessing the Device ID.	JScript: `var nDeviceID = Context.GetProperty("DeviceID");`
Accessing the IP address of the device.	JScript: `var sAddress = Context.GetProperty("Address");`

Accessing the device credentials.
Note All passwords are decrypted.
	JScript:
	`var sV1ReadCommunity = Context.GetProperty("CredSnmpV1:ReadCommunity");`
	`var sV1WriteCommunity = Context.GetProperty("CredSnmpV1:WriteCommunity");`
	`var sV2ReadCommunity = Context.GetProperty("CredSnmpV2:ReadCommunity");`
	`var sV2WriteCommunity = Context.GetProperty("CredSnmpV2:WriteCommunity");`
	`var sNTUsername = Context.GetProperty("CredWindows:DomainAndUserid");`
	`var sNTPassword =  Context.GetProperty("CredWindows:Password");`

Monitor	Code
Use WMI to see who is currently logged into a device.	You can set the monitor to be down if the logged-in user is not the expected user. VBScript: sComputer = Context.GetProperty("Address") nDeviceID = Context.GetProperty("DeviceID") 'Assuming ICMP is not blocked and there's a ping monitor on the device, we want to 'perform the actual check only if the Ping monitor is up. ConnectServer method of 'the SWbemLocator has a long time out so it would be good to avoid unnecessary tries. 'Please note: there's no particular polling order of active monitors on a device. 'During each polling cycle, it's possible that this monitor could be polled before 'Ping is polled. If the network connection just goes down but Ping is not polled yet, 'and therefore still has an up state, this active monitor will still do an actual 'check and experience a real down. But for the subsequent polls, it won't be doing a 'real check (ConnectServer won't be called) as Ping monitor has a down state, and this 'monitor will be assumed down. If IsPingUp(nDeviceID) = false Then Context.SetResult 1,"Actual check was not performed due to ping being down. Automatically set to down." Else sAdminName = Context.GetProperty("CredWindows:DomainAndUserid") sAdminPasswd = Context.GetProperty("CredWindows:Password") sLoginUser = GetCurrentLoginUser(sComputer, sAdminName, sAdminPasswd) sExpectedUser = "administrator" If Not IsNull(sLoginUser) Then If instr(1,sLoginUser, sExpectedUser,1) > 0 Then Context.SetResult 0,"Current login user is " & sLoginUser ElseIf sLoginUser = " " Then Context.SetResult 0,"No one is currently logged in." Else Context.SetResult 1,"an unexpected user " & sLoginUser & " has logged in " & sComputer End If End If End If 'Check if Ping monitor on the device specified by nDeviceID is up. 'If nDeviceID is not available as it's in the case during discovery, then assume 'ping is up. 'If ping monitor is not on the device, then assume it's up so the real check will be 'performed.

<table>
<tr><th>Monitor</th><th>Code</th></tr>
<tr><td>Use WMI to see who is currently logged into a device.
(continued)</td><td><pre>
Function IsPingUp(nDeviceID)
If nDeviceID > -1 Then
'get the Ping monitor up state.
sSqlGetUpState = "SELECT sStateName from PivotActiveMonitorTypeToDevice as P
join " & _
"ActiveMonitorType as A on P.nActiveMonitorTypeID=A.nActiveMonitorTypeID " &
_
"join MonitorState as M on P.nMonitorStateID = M.nMonitorStateID " & _
"where nDeviceID=" & nDeviceID & " and A.sMonitorTypeName='Ping' and " & _
                      " P.bRemoved=0"
Set oDBconn = Context.GetDB
Set oStateRS = CreateObject("ADODB.Recordset")
' oStateRS.ActiveConnection = oDBconn
' oStateRS.CursorType =3 'adOpenStatic cursorType
       oStateRS.Open sSqlGetUpState,oDBconn,3
'if recordset is empty then
If oStateRS.RecordCount = 1 Then
If instr(1,oStateRS("sStateName"),"up",1) > 0 Then
IsPingUp = true
  Else
     IsPingUP = false
  End If
Else
  'if there's no ping on the device, then just assume up, so regular check
will happen.
  IsPingUp= true
End If
oStateRS.Close
oDBconn.Close
Set oStateRS = Nothing
Set oDBconn = Nothing
Else
'assume up, since there's no device yet. It's for scanning during discovery.
 IsPingUP = true
End If
End Function
'Try to get the current login user name.
Function GetCurrentLoginUser(sComputer, sAdminName, sAdminPasswd)
GetCurrentLoginUser=Null
Set oSWbemLocator = CreateObject("WbemScripting.SWbemLocator")
On Error Resume Next
Set oSWbemServices = oSWbemLocator.ConnectServer _
  (sComputer, "root\cimv2",sAdminName,sAdminPasswd)
If Err.Number <> 0 Then
 Context.LogMessage("The 1st try to connect to " & sComputer & " failed. Err:"
& Err.Description)
 Err.Clear
      'If the specified user name and password for WMI connection failed, then
 'try to connect without user name and password. Can't specify user name
 'and password when connecting to local machine.
On Error Resume Next
 Set oSWbemServices = oSWbemLocator.ConnectServer(sComputer, "root\cimv2")
 If Err.Number <> 0 Then
    Err.Clear
</pre></td></tr>
</table>

<table>
<tr><th>Monitor</th><th>Code</th></tr>
<tr><td>Use WMI to see who is currently logged into a device.
(continued)</td><td><pre>
    On Error Resume Next
    Context.SetResult 1,"Failed to access " & sComputer & " " & _
"using username:" & sAdminName & " password." & " Err: " & Err.Description
    Exit Function
 End If
End If

Set colSWbemObjectSet = oSWbemServices.InstancesOf("Win32_ComputerSystem")

For Each oSWbemObject In colSWbemObjectSet
On Error Resume Next
 'Context.SetResult 0,"User Name: " & oSWbemObject.UserName & " at " &
sComputer
  sCurrentLoginUser = oSWbemObject.UserName
  Err.Clear

Next

If Cstr(sCurrentLoginUser) ="" Then

  GetCurrentLoginUser = " "
Else
GetCurrentLoginUser = sCurrentLoginUser
End If
Set oSWbemServices = Nothing
Set oSWbemLocator = Nothing
End Function
</pre></td></tr>
<tr><td>Use SNMP to monitor the total bandwidth utilization on an interface (in + out octets) by polling values of the interface MIB.</td><td>JScript:<pre>
// Settings for this monitor:

// the interface index ifIndex:
var nInterfaceIndex = 65540;
// this monitor will fail if the interface utilization goes above this current
ratio:
// current bandwidth / maxBandwidth > nMaxInterfaceUtilizationRatio
var nMaxInterfaceUtilizationRatio = 0.7; // Set to 70%
// Create an SNMP object, that will poll the device.
var oSnmpRqst = new ActiveXObject("CoreAsp.SnmpRqst");
// Get the device ID
var nDeviceID = Context.GetProperty("DeviceID");
// This function polls the device returns the ifSpeed of the inteface indexed
by nIfIndex.
// ifSpeed is in bits per second.
function getIfSpeed(nIfIndex)
{
</pre></td></tr>
</table>

Monitor	Code

Use SNMP to monitor the total bandwidth utilization on an interface (in + out octets) by polling values of the interface MIB.
(continued)
	var oResult = oSnmpRqst.Initialize(nDeviceID);
	if(oResult.Failed)
	{
	    return null;
	}
	return parseInt(SnmpGet("1.3.6.1.2.1.2.2.1.5." + nIfIndex)); // ifSpeed
	}
	// Function to get SNMP ifInOctets for the interface indexed by nIfIndex (in bytes).
	// Returns the value polled upon success, null in case of failure.
	function getInOctets(nIfIndex)
	{
	var oResult = oSnmpRqst.Initialize(nDeviceID);
	if(oResult.Failed)
	{
	    return null;
	}
	return parseInt(SnmpGet("1.3.6.1.2.1.2.2.1.10." + nIfIndex)); // inOctets
	}
	// Function to get SNMP ifOutOctets for the interface indexed by nIfIndex (in bytes).
	// Returns the value polled upon success, null in case of failure.
	function getOutOctets(nIfIndex)
	{
	var oResult = oSnmpRqst.Initialize(nDeviceID);
	if(oResult.Failed)
	{
	   return null;
	}
	return parseInt(SnmpGet("1.3.6.1.2.1.2.2.1.16." + nIfIndex)); // outOctets
	}
	// Helper function to get a specific SNMP object (OID in sOid).
	// Returns the value polled upon success, null in case of failure.
	function SnmpGet(sOid)
	{
	var oResult = oSnmpRqst.Get(sOid);
	if(oResult.Failed)
	{
	   return null;
	}
	else
	{
	return oResult.GetPayload;
	}
	}
	// Get the current date. It will be used as a reference date for the SNMP polls.
	var oDate = new Date();
	var nPollDate = parseInt(oDate.getTime()); // get the date in millisec in an integer.
	// Do the actual polling:
	var nInOctets = getInOctets(nInterfaceIndex);

<table>
<tr><th>Monitor</th><th>Code</th></tr>
<tr><td>

Use SNMP to monitor the total bandwidth utilization on an interface (in + out octets) by polling values of the interface MIB.

(continued)

</td><td>

```
var nOutOctets = getOutOctets(nInterfaceIndex);
var nIfSpeed = getIfSpeed(nInterfaceIndex);
if (nInOctets == null || nOutOctets == null ||  nIfSpeed == null)
{
Context.SetResult(1, "Failure to poll this device.");
}
else
{
var nTotalOctets = nInOctets + nOutOctets;
// Retrieve the octets value and date of the last poll saved in a context
variable:
var nInOutOctetsMonitorPreviousPolledValue =
Context.GetProperty("nInOutOctetsMonitorPreviousPolledValue");
var nInOutOctetsMonitorPreviousPollDate =
Context.GetProperty("nInOutOctetsMonitorPreviousPollDate");
if (nInOutOctetsMonitorPreviousPolledValue == null ||
nInOutOctetsMonitorPreviousPollDate
== null)
{
    // the context variable has never been set, this is the first time we are
polling.
    Context.LogMessage("This monitor requires two polls.");
    Context.SetResult(0, "success");
}
else
{
// compute the bandwidth that was used between this poll and the previous poll
var nIntervalSec =  (nPollDate - nInOutOctetsMonitorPreviousPollDate)/1000;
// time since
    last poll in seconds
var nCurrentBps = (nTotalOctets - nInOutOctetsMonitorPreviousPolledValue) *
8 /
nIntervalSec;
Context.LogMessage( "total octets for interface " + nInterfaceIndex + " = "
+ nTotalOctets)
;
Context.LogMessage( "previous value = " +
nInOutOctetsMonitorPreviousPolledValue);
Context.LogMessage("difference: " + (nTotalOctets -
nInOutOctetsMonitorPreviousPolledValue)
+ " bytes");
Context.LogMessage("Interface Speed: " + nIfSpeed + "bps");
Context.LogMessage("time elapsed since last poll: " + nIntervalSec  + "s");
Context.LogMessage("Current Bandwidth utilization: "+ nCurrentBps + "bps");
if (nCurrentBps/nIfSpeed > nMaxInterfaceUtilizationRatio)
{
   Context.SetResult(1, "Failure: bandwidth used on this interface " +
nCurrentBps + "bps
   / total available: " + nIfSpeed + "bps is above the specified ratio: " +

   nMaxInterfaceUtilizationRatio);
   }
```

</td></tr>
</table>

Monitor	Code

Use SNMP to monitor the total bandwidth utilization on an interface (in + out octets) by polling values of the interface MIB.
(continued)
	`else`
	`{`
	`Context.SetResult(0, "Success");`
	`}`
	`}`
	`// Save this poll information in the context variables:`
	`Context.PutProperty("nInOutOctetsMonitorPreviousPolledValue", nTotalOctets)`
	`Context.PutProperty("nInOutOctetsMonitorPreviousPollDate", nPollDate);`
	`}`

Monitoring an SNMP agent running on a nonstandard port (161).	JScript:
	`var nSNMPPort = 1234; // change this value to the port your agent is running on`
	`var oSnmpRqst =  new ActiveXObject("CoreAsp.SnmpRqst");`
	`// Get the device ID`
	`var nDeviceID = Context.GetProperty("DeviceID");`
	`// Initialize the SNMP request object`
	`var oResult = oSnmpRqst.Initialize(nDeviceID);`
	`if(oResult.Failed)`
	`{`
	`Context.SetResult(1, oResult.GetPayload);`
	`}`
	`else`
	`{`
	`// Set the request destination port.`
	`var oResult = oSnmpRqst.SetPort(nSNMPPort);`
	`// Get sysDescr.`
	`var oResult = oSnmpRqst.Get("1.3.6.1.2.1.1.1.0");`
	`if (oResult.Failed)`
	`{`
	`Context.SetResult(1, "Failed to poll device using port " + nSNMPPort + ".`
	`Error=" +`
	`oResult.GetPayload);`
	`}`
	`else`
	`{`
	`Context.SetResult(0, "SUCCESS. Detected an SNMP agent running on port "`
	`+ nSNMPPort );`
	`}`
	`}`

Using the Active Script Monitor Context Object

The context object is available to the script programmer when scripts are executing. It delivers context aspects of the device that it is operating upon. All methods and properties are retrieved using the "Context" namespace.

We have provided several code samples for you to use to create active script monitors for your devices.

Methods	Code

LogMessage(sText);	Allows for a message to be written to the Cisco netManager debug log.
	For example,
	JScript:
	`Context.LogMessage( "Checking Monitor name using Context.GetProperty()");`
	VBScript:
	`Context.LogMessage "Checking Address using Context.GetProperty()"`

PutProperty(sPropertyName);	Allows you to store a value in the INMSerialize object. This value is retained across polls.
	For example,
	JScript:
	`var nCount = parselnt(nNum) +1;`
	`Context.PutProperty("MyNumeric",nCount);`

SetResult(nCode, sText);	Allows for a result code and result message to be set. This is how you can tell the Cisco netManager system if the monitor succeeded or not.
	Note Every script should have a result, otherwise it will report back positively.
	For example,
	JScript:
	`Context.SetResult(0, "    Everything is OK"); //Success`
	`Context.SetResult(1, "    Really big big error"); //Failure`
	VBScript:
	`Context.SetResult 1, "    Really big big error"`

Properties

```
GetProperty(sPropertyName);
```

This property offers access to many device-specific aspects. You obtain access to these items using the names listed. These names are case sensitive.

Names	Description
ActiveMonitorTypeName	The active monitor display name
Address	The IP address of the device
DeviceID	The device ID
Mode	1 = Doing discovery 2 = Polling 3 = Test
ActiveMonitorTypeID	The active monitor's type ID
CredSnmpV1:ReadCommunity	SNMPv1 read community
CredSnmpV1:WriteCommunity	SNMPv1 write community
CredSnmpV2:ReadCommunity	SNMPv2 read community
CredSnmpV2:WriteCommunity	SNMPv2 write community
CredWindows:DomainAndUserid	Windows NT domain and user ID
CredWindows:Password	Windows NT password

Properties Example1 1

JScript:

```
var sAddress = Context.GetProperty("Address");
var sReadCommunity = Context.GetProperty("CredSnmpV1:ReadCommunity");
var nDeviceID = Context.GetProperty("DeviceID");
```

JScript:

```
//Sending log message to the Cisco netManager Event Viewer
Context.LogMessage ( "Checking Mode flag");
var nFlag = Context.GetProperty("Mode");

if (nFlag == 1)
{
Context.LogMessage ("Doing a discovery");
}
else if (nFlag == 2)
{
Context.LogMessage ("Doing a poll");
}
else if (nFlag == 3)
{
```

```
Context.LogMessage ("Must be just a test.");
}
else
{
Context.LogMessage ("Do not know the mode.");
}
//Set the result code of the check (0=Success, 1=Error)
Context.SetResult (0, "No error");
(GetDB);
```

This property returns an open connection to the Cisco netManager database.

Properties Example 2

This example gets the Open connection and reads some values out of the Cisco netManager "Device" table using the deviceID context.

```
var oDb = Context.GetDB;
if (null == oDb)
{
Context.SetResult( 1, "  Problem creating the PRO DB object");
}
else
{
var oRs = new ActiveXObject("ADODB.Recordset");
// Get the device ID
var nDeviceID = Context.GetProperty("DeviceID");
var sSql = "SELECT * from Device WHERE nDeviceID = " + nDeviceID;
oRs = oDb.Execute(sSql);
if ( !oRs.EOF )
{
   var sDisplay;
   sDisplay = "" + oRs("sDisplayName");
   Context.LogMessage("Display Name=" + sDisplay);
   sDisplay = "" + oRs("nWorstStateID");
   Context.LogMessage("WorstStateID=" + sDisplay);
   sDisplay = "" + oRs("sNote");
   Context.LogMessage("Note=" + sDisplay);
   sDisplay = "" + oRs("sStatus");
   Context.LogMessage("Status=" + sDisplay);
}
Context.SetResult( 0, "   Ok");
}
```