CHAPTER 1

The WhatsConfigured Custom Script Language

In This Chapter

About the WhatsConfigured Custom Script Language

WhatsConfigured users can write custom scripts that log in to devices through Telnet or SSH and run CLI commands on their devices. Scripts can be used to configure devices or to capture information about them in the WhatsConfigured database. For example, the following script uses Cisco IOS commands to capture a Cisco device's running configuration in the Network Performance Monitor database under the "running-config" key.

```
#

# Cisco IOS Backup Running Configuration

#

# login to the device

@login

#enter privileged mode

@enable

# display the running configuration of the device and capture it in the
WUG database

[running-config] show run

# logout from the device

@logout
```

The WhatsConfigured custom script language is relatively simple and consists primarily of command-line interface (CLI) commands. The language is not meant to be a full-featured scripting language, such as JavaScript or VBScript, but rather it is kept simple so that is

accessible to all levels of WhatsConfigured users, including those with minimal programming skills. In order to meet the standards of this target audience, the language contains no constructs for looping, complex branching, or creating subroutines; it only supports simple sequences of commands.

The custom script language has three possible elements:

- Comments
- Variables
- Commands

Each of these elements is explained in detail in the following sections.

CHAPTER 2

Using WhatsConfigured Comments

In This Chapter

About WhatsConfigured comments

In a script, you have the option to insert details or notes about the script. These notes and details are entered as comments, or lines having # as their first non-whitespace character. Comments are ignored by the script interpreter.

Note: A # character is interpreted as the beginning of a comment only if it is the first non-whitespace character on a line. If the # appears later in the line, it has no special significance.

Examples

```
# This is a comment

#         This is also a comment

123 # This is not a comment because '#' is not the first non-whitespace
character in the line
```

CHAPTER 3

Using WhatsConfigured Variables

In This Chapter

About variables

Variables are useful for giving names to values referenced in a script, especially values that are referenced multiple times. For example,

```
CommandTerminator = "\r\n"
TFTPServerAddress = 192.168.10.50
TransferFileName= startup-config.txt
@login
@write "copy tftp start"
@write $(CommandTerminator)
@write "$(TFTPServerAddress)"
@write "(TransferFileName)"
@write $(CommandTerminator)
```

Variable definitions

A variable definition must appear on a line by itself, in the following form:

Name = Value

In the example above, Name is the variable's title, and Value is the variable's value.

Variable names must begin with an alphabetic character or an underscore (a-z, A-Z, _), and subsequent characters can be any alphanumeric character or an underscore (a-z, A-Z, 0-9, _).

Note: Spaces are not allowed in variable names.

The variable's value consists of all text on the right side of = with leading and trailing whitespace removed. For example,

```
FirstUSPresident =        The Honorable George Washington
```

The example above defines a variable named "FirstUSPresident" with the value "The Honorable George Washington".

A variable's value can be referenced anywhere in the script after the variable is defined. A variable reference consists of '$' immediately followed by the variable's name in parentheses, as shown below.

```
$(FirstUSPresident)
```

A variable reference is replaced by the variable's value. If the variable is defined multiple times in the script, the most recent definition is used. In the example above, the variable reference "`$(FirstUSPresident)`" would be replaced by "The Honorable George Washington".

Accessing protocol settings

When a WhatsConfigured script runs, it executes against a particular device. The script uses the device's SSH or Telnet credentials to login to the device. Sometimes it is necessary for a script to directly access the protocol settings being used in a set of credentials. The protocol settings can be accessed through the variables listed in the following table.

Note: The values of these variables are read-only and cannot be modified by scripts, though scripts are free to reference their values.

Name	Description	Example
Settings.UserName	The SSH or Telnet username.	admin
Settings.Password	The SSH or Telnet password.	secret
Settings.PrivilegedPassword	The enable or privileged mode password.	supersecret

Using variables with WhatsConfigured templates

Templates allow network admins to automatically push device configurations to devices of the same type by replacing device-specific (IP address, hostname) information with variables, saving them time and reducing the possibility of error from one manual device configuration to another.

Note: The values of these variables are read-only and cannot be modified by scripts, though scripts are free to reference their values.

Name	Description
DateTime	The current system date and time in the format yyMMdd-HHmmss.
Device.IPv4Address	The device's IPv4 address.
Device.IPv6Address	The device's IPv6 address.
Device.IpAddress	The device's network IP address.
Device.SystemName	The device's hostname.
FileCaptureKey	The presence of this variable in a script indicates that the device configuration specified with the variable should be captured from the file system as part of the transfer task.
FileTransferMethod	Determines the file transfer method to be used for the file transfer. Can be: SCP_SERVER, SCP_Client, SFTP_SERVER, SFTP_CLIENT, TFTP_SERVER.
ReadTimeout	Specifies how long WhatsConfigured should wait to read expected output after a command is issued.
ScpClientDirectory	The SCP client directory.
ScpServerAddress	The IP address of the SCP server.
ScpServerPassword	The password of the SCP server.
ScpServerPort	The port of the SCP server.
ScpServerUserName	The username required to connect to the SCP server.
SftpClientDirectory	The SFTP client directory.
SftpServerAddress	The IP address of the SFTP server.
SftpServerPassword	The password of the SFTP server.
SftpServerPort	The port of the SFTP server.
SftpServerUserName	The username required to connect to the SFTP server.
TransferFileDirectory	The directory of the specified transfer method (FileTransferMethod); either SCP_SERVER, SCP_Client, SFTP_SERVER, SFTP_CLIENT, TFTP_SERVER. If an invalid FileTransferMethod is entered, this defaults to the TFTP_SERVER directory.
TransferFileName	The name of the device and the time of the file transfer.

Using reserved WhatsConfigured variable names

Script authors can use any names they want for their variables. However, the variables listed below are used internally by WhatsConfigured. As a general rule, script authors should avoid using these variable names in their scripts.

- AddCommunity RO
- AddCommunity RW
- AddPassword
- AddPrivilegedPassword
- AddReadOnly
- AddReadWrite
- AddUserName
- CommandPrompt
- CommandTerminator
- FileTransferMethod
- LoginTerminator
- MorePrompt
- MoreResponse
- Password
- PasswordPrompt
- PrivilegedPassword
- RemoveCommunity RO
- RemoveCommunity RW
- RemovePassword
- RemovePrivilegedPassword
- RemoveReadOnly
- RemoveReadWrite
- RemoveUserName
- TFTPServerAddress
- TransferFileName
- TransferFileDirectory
- UserName
- UserNamePrompt

Occassionally, a script may need to re-define one or more of these variables to affect the internal operation of WhatsConfigured commands. The section on WhatsConfigured commands describes the meanings and uses of these variables, and how scripts can re-define them to modify the behavior of WhatsConfigured commands.

CHAPTER 4

Using WhatsConfigured Commands

In This Chapter

About commands

Beyond commands and variables definitions, the other lines in a script contain the commands to be executed by the script.

Examples

```
@login

@enable

config t

line vty 0 4

login local

exit

username $(NewUserName) password $(NewPassword)

exit
```

```
@logout
```

About basic WhatsConfigured command syntax

There are two types of commands that can be included in a WhatsConfigured custom script:

- WhatsConfigured commands
- Device commands

WhatsConfigured commands are executed by WhatsConfigured itself. Device commands are executed by the device. WhatsConfigured commands begin with `@` to distinguish them from device commands. Any command whose text begins with `@` is a WhatsConfigured command, while any other command is a device command. In the previous example script, the `@login` and `@write` commands are WhatsConfigured commands, while all other commands are device commands.

WhatsConfigured defines the following commands:

- `@login`
- `@connect`
- `@write`
- `@read`
- `@read-more`
- `@logout`
- `@if`
- `@endif`
- `@scp-client-transfer`
- `@sftp-client-transfer`

The syntax for each of these commands is defined by WhatsConfigured. In contrast, Device commands are written using the native CLI commands supported by the device (IOS or CasOS commands for Cisco devices, Linux commands for Linux devices, etc.) These commands can use whatever syntax is required by the device's CLI command set.

In its simplest form, a command is just a string specifying the name of a command along with any parameters it requires. For example, the following script contains two simple commands:

```
@ login

username $(NewUserName) password $(NewPassword)
```

About strings and regular expressions in WhatsConfigured

WhatsConfigured commands make use of two specific types of values, strings and expressions.

Strings are used to represent literal text values; string values are sequences of characters delimited by double quotes, such as:

"Four score and seven years ago"

Escape sequences (used to define special characters within strings) may be any of the following:

Escape sequence	Represents
\0	Null character
\'	Single quote
\"	Double quote
\?	Literal question mark
\\	Backslash
\a	Bell alert (audible bell)
\b	Backspace
\f	Formfeed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Additionally, the \x escape sequence can be used to include arbitrary characters in strings, including unprintable and control characters. \x should be followed by one to four hexadecimal digits which specify the value of the desired character. For example,

```
"This is Control-S: \x13"
```

Regular expressions are used for matching patterns in the output of script commands. Regular expression values are .NET regular expression strings delimited by forward slashes. For example, the following regular expression might be used to match the command prompt on a particular device (i.e., one or more characters followed by > or #):

```
/.+(>|#)/
```

Because forward slashes are used to delimit regular expression values, including a forward slash as part of the regular expression itself, requires the use of the // escape sequence. For example, the following regular expression matches one or more characters followed by a forward slash followed by one or more characters followed by #:

```
/.+//.+#/
```

Note: All regular expression matching is case-insensitive.

About "$"

Within WhatsConfigured scripts the dollar sign, '$', is a reserved character required for use in variable references. If a script requires the use of a dollar sign outside of the variable reference it may be escaped with an additional dollar sign.

For example to write the Password 'pa$$word' you would escape the two dollar signs with additional dollar signs: 'pa$$$$word'

Note: To avoid erroneous dollar sign variable references, WhatsConfigured allows a single unescaped dollar sign when not followed by an open parentheses "`$(`". A line containing "`thi$`" would pass without needing to be escaped. Whereas the line "`thi$(`" would require the dollar sign be escaped "`thi$$(`".

Use the table below as an illustration of how many $'s you must enter in WhatsConfigured to achieve the appropriate number of $'s in a script command.

To achieve x # of $'s	Enter this many $'s in the code
$	$$
$$	$$$ or $$$$
$$$	$$$$$ (5) or $$$$$$ (6)
$$$$	$$$$$$$ (7) or $$$$$$$$ (8)

Storing WhatsConfigured command output in the Network Performance Monitor database

Most WhatsConfigured script commands return the output received from the device when the command was executed. For example, the `show run` command on Cisco devices displays the running configuration of the device. Or, on a Linux device, the `ls-al` command displays the contents of the current working directory. It is sometimes desirable to capture the output of the WhatsConfigured command and store it in the Network Performance Monitor database. To facilitate the storage of command output in the Network Performance Monitor database, a command can be preceded by a KEY which specifies the key under which the command's output should be restored in the Network Performance Monitor database. For example, the following command stores its output under the `running-config` key in the Network Performance Monitor database.

```
[running-config] show run
```

This means, execute the `show run` command and store its output in the Network Performance Monitor database under the `running-config` key.

Note: Key names can include dashes, underscores, and alphanumeric characters (-,_,a-z, A-Z). However, spaces are not allowed in key names.

Editing WhatsConfigured command output

Before storing a command's output in the Network Performance Monitor database, it is sometimes desirable to edit the output. For example, a command might place empty lines at the beginning or end of its output, and you may want to remove these empty lines before putting the output into the database. For situations like this, several operators are provided for editing command output. These operators are specified as part of the command's KEY. For example, the following command specifies that 4 lines should be trimmed from the output of the `ls –al` command before the output is stored in the Network Performance Monitor database under the key, `file-list`.

```
[file-list, trim-start-lines = "4"]   ls -al
```

The following output editing operators are provided:

Name	Value	Meaning	Example
trim-start-lines	Integer	Trim the first N lines from the commands output	trim-start-lines="1"
trim-end-lines	Integer	Trim the last N lines from the commands output	trim-end-lines="1"
trim-start	String or regular expression	Trim all output before and including the first match of the specified string or regular expression	trim-start="#\n#\n#\n"
trim-end	String or regular expression	Trim all output including and after the last match of the specified string or regular expression	trim-end="#\n#\n#\n"
trim-before	String or regular expression	Trim all output before the first match of the specified string or regular expression	trim-before="!"
trim-after	String or regular expression	Trim all output after the last match of the specified string or regular expression	trim-after"!"
remove-lines	String or regular expression	Remove all lines that match the specified string or regular expression	remove-lines=/system time.+/
remove-pattern	String or regular expression	Remove all words that match the specified string or pattern	remove-pattern=/\d?:/

If multiple editing operators are used in the same command, they are applied in the order shown in the previous table.

```
[file-list, trim-start-lines = "4", trim-end="\n\n\n"]   ls -al
```

Using WhatsConfigured commands with queries

Some Device commands require users to answer a question before the command is executed. For example, the `enable` command on Cisco devices queries the user for a password before executing the command. For this reason, a command can optionally specify a QUERY which specifies the question asked by the device and the answer that should be given to the question. The QUERY is specified after the command within curly braces. For example,

```
shutdown { "Are you sure? ", "Y" }
```

The first value inside the curly braces is a String or Regular Expression describing the query prompt displayed by the device. The second value inside the curly braces is a String specifying the query response that should be entered in response to the query prompt. When the script interpreter executes this command, it will first send `shutdown` to the device. Next, it will wait until it receives the "Are you sure? " query prompt. Then, it will send `Y` to the device as the query response. Finally, the device will execute the command.

Note: Only Device commands can have a QUERY, with the exception of `@logtout`. WhatsConfigured commands do not need a QUERY, and, in fact, may not have one, with the exception of `@logout`.

For example,

```
enable { $(PasswordPrompt), "$(Settings.PrivilegedPassword)" }
```

About WhatsConfigured command layout

The general format of a script command is:

KEY COMMAND QUERY

For example,

```
[last-words]   shutdown   { "Are you sure? ", "Y" }
```

As previously explained, KEY specifies the key to use when storing the command's output in the Network Performance Monitor database, and possibly operations for trimming the command output. COMMAND is the text for the command itself. QUERY specifies the query prompt and query response for commands that ask a question. COMMAND is required, while KEY and QUERY are optional.

Since commands can become long, it is legal to put the KEY, COMMAND, and QUERY parts of a command on different lines. For example, the following commands are equivalent:

```
[last-words]  shutdown  { "Are you sure? ", "Y" }
```

```
[last-words]

shutdown

{ "Are you sure? ", "Y" }

[last-words]

shutdown  { "Are you sure? ", "Y" }

[last-words] shutdown

{ "Are you sure? ", "Y" }
```

While the KEY, COMMAND, and QUERY can be on different lines from each other, each of these individual elements must start and end on the same line (i.e., they cannot span multiple lines). For example, the following commands are not valid:

```
[

last-words

]  shutdown  { "Are you sure? ", "Y" }

[last-words] shut

down  { "Are you sure? ", "Y" }

[last-words]

shutdown

{

"Are you sure? ",

"Y"

}
```

WhatsConfigured script variables affecting command execution

When running a script, WhatsConfigured defines several script variables that contain information necessary to execute the script's commands. For example, the `CommandPrompt` variable contains a pattern (i.e., string or regular expression) that describes the command prompt string used by the device. This pattern is used to detect when the device is prompting for a command. Several other variables are also defined. A complete list of all script variables affecting command execution are listed in the following table.

WhatsConfigured's assigns default values to each of these variables. If a script author wants to override WhatsConfigured's default behavior, he or she may do so by re-defining one or more of these variables. For example, if a script wants to override the command prompt pattern used to run the script, it can re-define the `CommandPrompt` variable to contain the pattern of choice.

Name	Value	Meaning	Example
UserNamePrompt	String or regular expression	Pattern describing the username prompt displayed by the device when a user logs in	"login as:"
PasswordPrompt	String or regular expression	Pattern describing the password prompt displayed by the device when a user logs in	"password:"
CommandPrompt	String or regular expression	Pattern describing the command prompt displayed by the device when prompting the user for a command	/.+(#|>) /
MorePrompt	String or regular expression	Pattern describing the "more" prompt displayed by the device when displaying paged output	/--More--|--More--/
MoreResponse	String	String to be entered in response to a "more" prompt	""
LoginTerminator	String	Line termination sequence to be used with logging in	"\r\n"
CommandTerminator	String	Line termination sequence to be used when executing commands	"\n"

About WhatsConfigured command types

There are two types of commands in a WhatsConfigured script: device commands and WhatsConfigured commands. Device commands can be any CLI command supported by a device, and are executed by the device. WhatsConfigured commands start with '@' and are executed by WhatsConfigured rather than by the device. The following sections describe the available commands and explain when and how to use them. Most scripts use a combination of Device commands and WhatsConfigured commands, although it is possible to write scripts using only WhatsConfigured commands.

@login

Typically, the first step in any WhatsConfigured script is to login to the device. This is typically done with the WhatsConfigured `@login` command. The `@login` command can be used to login to devices that use a traditional user-name/password login procedure that works as follows:

1. The device prompts the user for their user name
2. The user enters their user name
3. The device prompts the user for their password

4 The user enters their password
5 If login is successful, the device displays a command prompt and waits for the user to run commands

The `@login` command has no parameters, and is invoked as follows:

```
@login
```

When the `@login` command is executed, it does the following:

1 When it detects `UserNamePrompt`, it sends `Settings.UserName` to the device followed by `LoginTerminator`.
2 When it detects `PasswordPrompt`, it sends Settings.Password to the device followed by `LoginTerminator`.
3 When it detects `MorePrompt`, it enters `MoreResponse`.
4 After entering the user name and password, if `@login` detects `CommandPrompt`, it assumes that login was successful. Otherwise, it assumes that login failed.

If at any time the device's output stalls for more than `Settings.ReadTimeout` seconds, it is assumed that something is wrong, and the script returns failure.

@enable

Many device configuration tasks require a script to enter a privileged mode in order to execute the necessary device commands. On many devices, privileged mode is entered using the `enable` command. Typically, running the `enable` command on a device requires the user to enter a user name and/or password. For devices that implement this style of `enable` command, scripts can use the WhatsConfigured `@enable` command to easily enter privileged mode. The `@enable` command has no parameters, and is invoked as follows:

```
@enable
```

When the `@enable` command is executed, it does the following:

1 It sends `enable` to the device followed by `CommandTerminator`.
2 If it detects `UserNamePrompt`, it sends `Settings.UserName` to the device followed by `CommandTerminator`.
3 If it detects `PasswordPrompt`, it sends `Settings.PrivilegedPassword` to the device followed by `CommandTerminator`. If `Settings.PrivilegedPassword` is empty, it uses `Settings.Password` instead.
4 After entering the user name and password (if necessary), if `@enable` detects `CommandPrompt`, it assumes that enable was successful. Otherwise, it assumes that enable failed.

If at any time the device's output stalls for more than `Settings.ReadTimeout` seconds, it is assumed that something is wrong, and the script returns failure.

@logout

The last step of any WhatsConfigured script is typically to logout of the device. This is preferably done with the WhatsConfigured @logout command.

The @logout command writes out a logout command (the command defaults to exit) to the device with optional parameters to specify the command written and to specify expected device queries. After the command is written, WhatsConfigured waits to read the device's output.

The @logout command takes the following form:

```
@logout <optional device logout word><optional query>
```

Examples

Command	Output
`@logout`	Writes `exit` to the device.
`@logout "logout"`	Writes `logout` to the device.
`@logout "logout" {"Are you sure?", "Y"}`	Writes `logout` to the device, then waits to read `"Are you sure?"` When `"Are you sure?"` is found, WhatsConfigured writes `"Y"` to the device.

Device commands

After invoking the @login command (and possibly @enable as well), most scripts contain a sequence of device commands that are sent to the device for execution. A typical device command is shown below:

Device commands are executed as follows:

```
[last-words]   shutdown   { "Are you sure? ", "Y" }
```

The script sends the command text to the device. It terminates the command with CommandTerminator.

If the command has a query, the device returns the query to the script. When it detects the query prompt, the script sends the query response to the device.

Next, the device executes the commands, and sends its output back to the script.

If the command's output is long enough to result in more prompts, when the script detects a MorePrompt, it sends MoreResponse to the device.

The script consumes the command's output until it detects CommandPrompt, at which point it assumes that the command's output is complete.

If at any time the device's output stalls for more than Settings.ReadTimeout, it is assumed that something is wrong, and the script returns failure.

If the command succeeds, and it has a KEY, its output is saved in the Network Performance Monitor database.

The following script is typical:

```
@login
```

```
 @enable { "password: ", "$(Settings.PrivilegedPassword)" }

[running-config] show run

[-] logout
```

This script first logs in with the @login command, then enters privileged mode with the @enable command.

Next, the script sends the `show run` command to the device. The output of this command is saved in the Network Performance Monitor database under the `running-config` key.

Finally, the script sends the `logout` command to the device, at which point the device closes the network connection.

The [-] key on the `logout` command tells WhatsConfigured not to expect any output from the command, because the command causes the device to close the network connection. Typically, receiving no output from a command indicates failure, but in the case of `exit` or `logout` commands (or any other command that closes the connection), a lack of output does not indicate failure. Script authors can use the [-] key to indicate such commands and prevent WhatsConfigured from returning failure when the device closes the connection. Alternatively, `@logout` can be used to close the connection without failure.

Low-level commands

Many scripts use only `@login`, `@enable`, and device commands to implement their functionality. However, some devices have non-standard Telnet or SSH interfaces that do not work with `@login` and device commands. For example, some devices have non-standard login procedures for which the `@login` command does not work. Other devices have menu-driven interfaces rather than a standard command-prompt-style interface. For whatever reason, if `@login`, `@enable`, or device commands do not work for a particular device, WhatsConfigured provides a set of low-level commands that can be used to interact with virtually any device, no matter how non-standard its interface might be.

The `@connect` command allows a script to precisely control the process of logging in to a device. The `@write` command allows a script to control exactly what input is sent to a device. The `@read` command allows a script to read output from a device and optionally store it in the Network Performance Monitor database.

@connect

The `@connect` command is an alternative to `@login` in cases where a script needs to precisely control the login process (e.g., in cases where `@login` doesn't work for a particular device). The `@connect` command connects to a device without trying to log in. After connecting to a device with @connect, if the device requires users to login, the script can control the login process precisely using the `@write` and `@read` commands, which are described later.

When calling `@connect`, scripts can specify one or more patterns (i.e., strings or regular expressions) that specify the output the script expects to receive from the device when it connects. These patterns are used by `@connect` to detect the end of the device output.

`@connect` assumes that device output is complete when either the output matches one of the specified patterns, or no new output is received from the device for `Settings.ReadTimeout` seconds. For example,

```
@connect "login as: ", "user name: "
```

When executed, `@connect` connects to the device, and reads whatever output comes back from the device. If the output matches one of the specified patterns, the command succeeds. If the connection attempt fails entirely, or the output received from the device does not match any of the specified patterns, the command fails (as well as the entire script). As with any other command, a KEY can be specified to capture the command's output in the Network Performance Monitor database, although one would rarely want to store the output of an `@connect` command.

If no patterns are specified (as shown below), `@connect` connects to the device, and returns whatever output comes back from the device. In this case, the command succeeds as long as a connection is successfully established with the device.

@connect-more

Some devices return paged output that requires `more` prompts when you initially connect to them. If a script needs to handle `more` prompts during the connection process, it can use the `@connect-more` command instead of `@connect`. `@connect-more` works just like `@connect`, except that it handles `more` prompts during the connection process, while `@connect` does not. Specifically, if `MorePrompt` is detected during the connection process, `@connect-more` sends `MoreResponse` to the device.

```
@connect-more "login as: ", "user name: "
```

@write

The `@write` command can be used to send a string of characters to the device. This command allows a script to precisely control what input is being sent to the device. For example, the following script sends the `show run` command to the device, followed by the `CommandTerminator` (typically `\n` or `\r\n`).

```
@write "show run"

@write $(CommandTerminator)
```

@read

The `@read` command can be used by scripts to read the output coming back from the device. Typically, a call to `@read` immediately follows a call to `@write`. When calling `@read`, scripts can specify one or more patterns (i.e., strings or regular expressions) to help `@read` detect the end of the device output. `@read` assumes that device output is complete when either: the output matches one of the specified patterns, or no new output is received from the device for `Settings.ReadTimeout` seconds. Often, the output ends with `CommandPrompt`, so "`@read $(CommandPrompt)`" is a common way to call `@read`. If desired, the output received from the device can be stored in the Network Performance Monitor database, as shown below:

```
@write "show run"

@write $(CommandTerminator)

[running-config, trim-end-lines = 1]  @read  $(CommandPrompt)
```

When executed, `@read` reads whatever output comes back from the device. If the output matches one of the specified patterns, the command succeeds. If the output received from the device does not match one of the specified patterns, the command fails (as well as the entire script).

If no patterns are specified (as shown below), `@read` returns whatever output comes back from the device. In this case, the command succeeds as long as the connection to the device is still open.

@read-more

Some devices return paged output that requires `more` prompts. If a script needs to handle `more` prompts during a read operation, it can use the `@read-more` command instead of `@read`. `@read-more` works just like `@read`, except that it handles `more` prompts during the reading process, while `@read` does not. Specifically, if `MorePrompt` is detected during the reading process, `@read-more` will send `MoreResponse` to the device.

CHAPTER 5

Script Examples

In This Chapter

Example Scripts

This example shows a typical script that uses `@login` to login to the device, uses `@enable` to enter privileged mode, and then executes several device commands.

```
@login

@enable

[running-config] show run

@logout
```

This example shows how to login to a device and run a command using only low-level WhatsConfigured commands:

```
@connect "login as: "

@write "$(Settings.UserName)"

@write $(LoginTerminator)

@read "password: "

@write "$(Settings.Password)"

@write $(LoginTerminator)

@read $(CommandPrompt)

@write "exit"

@write $(CommandTerminator)
```

This example shows how to combine high-level commands and low-level commands in the same script as above:

```
@login

@enable

@write "copy tftp start"

@write $(CommandTerminator)

@write "$(TFTPServerAddress)"

@write $(CommandTerminator)

@write "$(TransferFileName)"

@write $(CommandTerminator)

@write $(CommandTerminator)

@logout
```

Copyright notice

This document was published on Friday, March 04, 2016 at 11:22.