Command-Line Interface

This chapter provides information for understanding and using the Catalyst 6500 series switch SSL Services Module software using the command-line interface (CLI). The CLI for the Catalyst 6500 series switch SSL Services Module is based on the Cisco IOS CLI. For information about Cisco IOS commands that are not contained in this publication, refer to the current Cisco IOS documentation including:

- *Cisco IOS Release 12.2 Configuration Fundamentals Configuration Guide*
- *Cisco IOS Release 12.2 Command Reference*

This chapter includes the following sections:

Getting Help

To obtain a list of commands that are available for each command mode, enter a question mark (**?**) at the system prompt. You also can obtain a list of any command's associated keywords and arguments with the context-sensitive help feature.

Table 1-1 lists commands that you can enter to get help that is specific to a command mode, a command, a keyword, or an argument.

Table 1-1 Getting Help

Command	Purpose
*abbreviated-command-entry***?**	Obtain a list of commands that begin with a particular character string. (Do not leave a space between the command and question mark.)
abbreviated-command-entry<**Tab**>	Complete a partial command name.
?	List all commands available for a particular command mode.

Table 1-1 Getting Help (continued)

Command	Purpose
command **?**	List a command's associated keywords. Leave a space between the command and question mark.
command keyword **?**	List a keyword's associated arguments. Leave a space between the keyword and question mark.

This example shows how to obtain a list of commands that begin with a particular character string or complete a partial command name:

```
ssl-proxy# tu?
tunnel

simpson1-2# tu
```

This example shows how to list all commands available for a particular command mode:

```
ssl-proxy(config)# ?
Configure commands:
  aaa                         Authentication, Authorization and
                              Accountin
  access-list                 Add an access list entry
  alias                       Create command alias
  arp                         Set a static ARP entry
  async-bootp                 Modify system bootp parameters
  banner                      Define a login banner
  boot                        Modify system boot parameters
  bridge                      Bridge Group.
  buffers                     Adjust system buffer pool parameters
  cdp                         Global CDP configuration subcommands
  class-map                   Configure QoS Class Map
.
.
.
Output is truncated.
```

This example shows how to list a keyword's associated arguments:

```
ssl-proxy(config-if)# channel-group 1 mode ?
  auto       Enable PAgP only if a PAgP device is detected
  desirable  Enable PAgP unconditionally
  on         Enable Etherchannel only

ssl-proxy(config-if)#
```

How to Find Command Options

This section provides an example of how to display syntax for a command. The syntax can consist of optional or required keywords. To display keywords for a command, enter a question mark (**?**) at the configuration prompt or after entering part of a command followed by a space. The Catalyst 6500 series SSL Services Module software displays a list of available keywords along with a brief description of the keywords. For example, if you are in global configuration mode and want to see all the keywords for the **ssl-proxy** command, you enter **ssl-proxy ?**.

Table 1-2 shows examples of how you can use the question mark (**?**) to assist you in entering commands.

Table 1-2 How to Find Command Options

Command	Comment

`ssl-proxy> enable`
`Password: <password>`
`ssl-proxy#`
	Enter the **enable** command and password to access privileged EXEC commands.
	You are in privileged EXEC mode when the prompt changes to `ssl-proxy#`.

`ssl-proxy# configure terminal`
`Enter configuration commands, one per line. End with CNTL/Z.`
`ssl-proxy(config)#`
	Enter global configuration mode.
	You are in global configuration mode when the prompt changes to `ssl-proxy(config)#`.

`ssl-proxy(config)# crypto ca trustpoint trustpoint-label` `ssl-proxy(ca-trustpoint)#`	Enter the configuration submode. You are in the configuration submode when the prompt displays the submode, for example: `ssl-proxy(ca-trustpoint)#`.
`ssl-proxy(config)# interface` *type mod/port* `ssl-proxy(config-if)#`	From the global configuration mode, you can also enter the interface configuration mode by entering the **interface** global configuration command. You are in interface configuration mode when the prompt changes to `ssl-proxy(config-if)#`.

`ssl-proxy(config-if)# channel-group ?`
`  group  channel-group of the interface`
 	
`ssl-proxy(config-if)#channel-group`
	Enter the command that you want to configure for the controller. In this example, the **channel-group** command is used.
	Enter a **?** to display what you must enter next on the command line. In this example, you must enter the **group** keyword.
	Because a `<cr>` is not displayed, it indicates that you must enter more information to complete the command.

Table 1-2 How to Find Command Options (continued)

Command	Comment

`ssl-proxy(config-if)# channel-group group ?`
`  <1-256>  Channel group number`
 	
`ssl-proxy(config-if)#channel-group group`
	After you enter the **group** keyword, enter a **?** to display what you must enter next on the command line. In this example, you must enter a channel group number from 1 to 256.
	
	Because a <cr> is not displayed, it indicates that you must enter more information to complete the command.

`ssl-proxy(config-if)# channel-group 1 ?`
`  mode  Etherchannel Mode of the interface`
 	
`ssl-proxy(config-if)#`
	After you enter the channel group number, enter a **?** to display what you must enter next on the command line. In this example, you must enter the **mode** keyword.
	
	Because a <cr> is not displayed, it indicates that you must enter more information to complete the command.

`ssl-proxy(config-if)# channel-group 1 mode ?`
`  auto       Enable PAgP only if a PAgP device is detected`
`  desirable  Enable PAgP unconditionally`
`  on         Enable Etherchannel only`
 	
`ssl-proxy(config-if)#`
	After you enter the **mode** keyword, enter a **?** to display what you must enter next on the command line. In this example, you must enter the **auto**, **desirable**, or **on** keyword.
	
	Because a <cr> is not displayed, it indicates that you must enter more information to complete the command.

`ssl-proxy(config-if)# channel-group 1 mode auto ?`
`  <cr>`
 	
`ssl-proxy(config-if)#`
	In this example, the **auto** keyword is entered. After you enter the **auto** keyword, enter a **?** to display what you must enter next on the command line.
	
	Because a <cr> is displayed, it indicates that you can press **Return** to complete the command. If additional keywords are listed, you can enter more keywords or press **Return** to complete the command.

`ssl-proxy(config-if)# channel-group 1 mode auto` `ssl-proxy(config-if)#`	In this example, press **Return** to complete the command.

Understanding Command Modes

This section contains descriptions of the command modes for the Cisco IOS user interface.

Cisco IOS User Interface

The Cisco IOS user interface is divided into many different modes. The commands that are available to you depend on which mode you are currently in. You can obtain a list of commands that are available for each command mode by entering a question mark (**?**) at the system prompt.

When you start a session on the Catalyst 6500 series switch, you begin in user mode, often called EXEC mode. Only a limited subset of the commands are available in EXEC mode. In order to have access to all commands, you must enter privileged EXEC mode. Normally, you must enter a password to enter privileged EXEC mode. From privileged EXEC mode, you can enter any EXEC command or enter global configuration mode. Most EXEC commands are one-time commands, such as **show** commands, which show the current status of a given item, and **clear** commands, which clear counters or interfaces. The EXEC commands are not saved across reboots of the Catalyst 6500 series switch.

The configuration modes allow you to make changes to the running configuration. If you later save the configuration, these commands are stored across Catalyst 6500 series switch reboots. In order to get to the various configuration modes, you must start at global configuration mode where you can enter interface configuration mode, subinterface configuration mode, and a variety of protocol-specific modes.

ROM-monitor mode is a separate mode that is used when the Catalyst 6500 series switch cannot boot properly. If your Catalyst 6500 series switch or access server does not find a valid system image when it is booting, or if its configuration file is corrupted at startup, the system might enter ROM-monitor mode.

Table 1-3 provides a summary of the main command modes.

Table 1-3 Summary of Main Command Modes

Command Mode	Access Method	Prompt	Exit Method
User EXEC	Log in.	`ssl-proxy>`	Use the **logout** command.
Privileged EXEC	From user EXEC mode, enter the **enable** EXEC command.	`ssl-proxy#`	To exit to user EXEC mode, enter the **disable** command. To enter global configuration mode, enter the **configure terminal** privileged EXEC command.
Global configuration	From privileged EXEC mode, enter the **configure terminal** privileged EXEC command.	`ssl-proxy(config)#`	To exit to privileged EXEC mode, enter the **exit** or **end** command or press **Ctrl-Z**. To enter interface configuration mode, enter an **interface** configuration command.
Global configuration submode	From global configuration mode, enter a submode command.	`ssl-proxy(config-submode)#`	To exit to global configuration submode, enter the **exit** command.

Table 1-3 Summary of Main Command Modes (continued)

Command Mode	Access Method	Prompt	Exit Method
Interface configuration	From global configuration mode, enter by specifying an interface with an **interface** command.	`ssl-proxy(config-if)#`	To exit to global configuration mode, enter the **exit** command. To exit to privileged EXEC mode, enter the **exit** command or press **Ctrl-Z**. To enter subinterface configuration mode, specify a subinterface with the **interface** command.
Subinterface configuration	From interface configuration mode, specify a subinterface with an **interface** command.	`ssl-proxy(config-subinterface)#`	To exit to global configuration mode, enter the **exit** command. To enter privileged EXEC mode, enter the **end** command or press **Ctrl-Z**.
ROM monitor	From privileged EXEC mode, enter the **reload** EXEC command. Press the Break key during the first 60 seconds while the system is booting.	`Rommon>`	To exit ROM-monitor mode, you must reload the image by entering the **boot** command. If you use the **boot** command without specifying a file or any other boot instructions, the system boots from the default Flash image (the first image in onboard Flash memory). Otherwise, you can instruct the system to boot from a specific Flash image (using the **boot system flash** *filename* command).

For more information on command modes, refer to the "Using the Command Line Interface" chapter of the *Configuration Fundamentals Configuration Guide*.

Note You can issue EXEC-level Cisco IOS commands (such as **show**, **clear**, and **debug** commands) from within global configuration mode or other modes by issuing the **do** command followed by the EXEC command. See the **do** command for information on how to use this command.

Using the No and Default Forms of Commands

Almost every configuration command has a **no** form. In general, enter the **no** form to disable a function. Use the command without the keyword **no** to reenable a disabled function or to enable a function that is disabled by default. For example, IP routing is enabled by default. To disable IP routing, specify the **no ip routing** command and specify the **ip routing** command to reenable it. This publication provides the complete syntax for the configuration commands and describes what the **no** form of a command does.

Configuration commands can have a **default** form. The **default** form of a command returns the command setting to its default. Most commands are disabled by default, so the **default** form is the same as the **no** form. However, some commands are enabled by default and have variables set to certain default values. In these cases, the **default** form of the command enables the command and sets variables to their default values. This publication describes what the **default** form of a command does if the command is not the same as the **no** form.

Using the CLI String Search

The pattern in the command output is referred to as a string. The CLI string search feature allows you to search or filter any **show** or **more** command output and allows you to search and filter at --More-- prompts. This feature is useful when you need to sort though large amounts of output, or if you want to exclude output that you do not need to see.

With the search function, you can begin unfiltered output at the first line that contains a regular expression that you specify. You can then specify a maximum of one filter per command or start a new search from the --More-- prompt.

A regular expression is a pattern (a phrase, number, or more complex pattern) that software uses to match against **show** or **more** command output. Regular expressions are case sensitive and allow for complex matching requirements. Examples of simple regular expressions are Serial, misses, and 138. Examples of complex regular expressions are 00210..., (is), and [Oo]utput.

You can perform three types of filtering:

- Use the **begin** keyword to begin output with the line that contains a specified regular expression.
- Use the **include** keyword to include output lines that contain a specified regular expression.
- Use the **exclude** keyword to exclude output lines that contain a specified regular expression.

You can then search this filtered output at the --More-- prompts.

Note The CLI string search function does not allow you to search or filter backward through previous output; filtering cannot be specified using HTTP access to the CLI.

Regular Expressions

A regular expression can be a single character that matches the same single character in the command output or multiple characters that match the same multiple characters in the command output. This section describes how to create both single-character patterns and multiple-character patterns and how to create more complex regular expressions using multipliers, alternation, anchoring, and parentheses.

Single-Character Patterns

The simplest regular expression is a single character that matches the same single character in the command output. You can use any letter (A-Z, a-z) or digit (0-9) as a single-character pattern. You can also use other keyboard characters (such as ! or ~) as single-character patterns, but certain keyboard characters have special meaning when used in regular expressions. Table 1-4 lists the keyboard characters with special meaning.

Table 1-4 Characters with Special Meaning

Character	Special Meaning
.	Matches any single character, including white space.
*	Matches 0 or more sequences of the pattern.
+	Matches 1 or more sequences of the pattern.
?	Matches 0 or 1 occurrences of the pattern.

Table 1-4 Characters with Special Meaning (continued)

Character	Special Meaning
^	Matches the beginning of the string.
$	Matches the end of the string.
_ (underscore)	Matches a comma (,), left brace ({), right brace (}), left parenthesis (( ), right parenthesis ( )), the beginning of the string, the end of the string, or a space.

To enter these special characters as single-character patterns, remove the special meaning by preceding each character with a backslash (\). These examples are single-character patterns matching a dollar sign, an underscore, and a plus sign, respectively.

\$ _ \+

You can specify a range of single-character patterns to match against command output. For example, you can create a regular expression that matches a string containing one of the following letters: a, e, i, o, or u. One and only one of these characters must exist in the string for pattern matching to succeed. To specify a range of single-character patterns, enclose the single-character patterns in square brackets ([ ]). For example,

[aeiou]

matches any one of the five vowels of the lowercase alphabet, while

[abcdABCD]

matches any one of the first four letters of the lower- or uppercase alphabet.

You can simplify ranges by entering only the end points of the range separated by a dash (-). Simplify the previous range as follows:

[a-dA-D]

To add a dash as a single-character pattern in your range, include another dash and precede it with a backslash:

[a-dA-D\-]

You can also include a right square bracket (]) as a single-character pattern in your range. To do so, enter the following:

[a-dA-D\-\]]

The previous example matches any one of the first four letters of the lower- or uppercase alphabet, a dash, or a right square bracket.

You can reverse the matching of the range by including a caret (^) at the start of the range. This example matches any letter except the ones listed:

[^a-dqsv]

This example matches anything except a right square bracket (]) or the letter d:

[^\]d]

Multiple-Character Patterns

When creating regular expressions, you can also specify a pattern containing multiple characters. You create multiple-character regular expressions by joining letters, digits, or keyboard characters that do not have special meaning. For example, a4% is a multiple-character regular expression. Put a backslash in front of the keyboard characters that have special meaning when you want to remove their special meaning.

With multiple-character patterns, order is important. The regular expression a4% matches the character a followed by a 4 followed by a % sign. If the string does not have a4%, in that order, pattern matching fails. This multiple-character regular expression

a.

uses the special meaning of the period character to match the letter a followed by any single character. With this example, the strings ab, a!, or a2 are all valid matches for the regular expression.

You can remove the special meaning of the period character by putting a backslash in front of it. In the following expression

a\.

only the string a. matches this regular expression.

You can create a multiple-character regular expression containing all letters, all digits, all keyboard characters, or a combination of letters, digits, and other keyboard characters. These examples are all valid regular expressions:

telebit 3107 v32bis

Multipliers

You can create more complex regular expressions to match multiple occurrences of a specified regular expression by using some special characters with your single- and multiple-character patterns. Table 1-5 lists the special characters that specify “multiples” of a regular expression.

Table 1-5 Special Characters Used as Multipliers

Character	Description
*	Matches 0 or more single- or multiple-character patterns.
+	Matches 1 or more single- or multiple-character patterns.
?	Matches 0 or 1 occurrences of the single- or multiple-character patterns.

This example matches any number of occurrences of the letter a, including none:

a*

This pattern requires that at least one letter a in the string is matched:

a+

This pattern matches the string bb or bab:

ba?b

This string matches any number of asterisks (*):

******

To use multipliers with multiple-character patterns, you enclose the pattern in parentheses. In the following example, the pattern matches any number of the multiple-character string ab:

(ab)*

As a more complex example, this pattern matches one or more instances of alphanumeric pairs (but not none; that is, an empty string is not a match):

([A-Za-z][0-9])+

The order for matches using multipliers (*, +, or ?) is to put the longest construct first. Nested constructs are matched from outside to inside. Concatenated constructs are matched beginning at the left side of the construct. The regular expression matches A9b3, but not 9Ab3 because the letters are specified before the numbers.

Alternation

Alternation allows you to specify alternative patterns to match against a string. You separate the alternative patterns with a vertical bar (|). Exactly one of the alternatives can match the string. For example, the regular expression

codex | telebit

matches the string codex or the string telebit, but not both codex and telebit.

Anchoring

You can match a regular expression pattern against the beginning or the end of the string. That is, you can specify that the beginning or end of a string contains a specific pattern. You "anchor" these regular expressions to a portion of the string using the special characters shown in Table 1-6.

Table 1-6 Special Characters Used for Anchoring

Character	Description
^	Matches the beginning of the string.
$	Matches the end of the string.

This regular expression matches a string only if the string starts with abcd:

^abcd

In contrast, this expression is in a range that matches any single letter, as long as it is not the letters a, b, c, or d:

[^abcd]

With this example, the regular expression matches a string that ends with .12:

$\.12

Contrast these anchoring characters with the special character underscore (_). The underscore matches the beginning of a string (^), the end of a string ($), parentheses ( ), space ( ), braces { }, comma (,), or underscore (_). With the underscore character, you can specify that a pattern exist anywhere in the string.

For example,

_1300_

matches any string that has 1300 somewhere in the string. The string's 1300 can be preceded by or end with a space, brace, comma, or underscore. For example,

{1300_

matches the regular expression, but 21300 and 13000 do not.

Using the underscore character, you can replace long regular expression lists, such as the following:

^1300$ ^1300(space) (space)1300 {1300, ,1300, {1300} ,1300, (1300

with

_1300_

Parentheses for Recall

As shown in the "Multipliers" section on page 1-9, you use parentheses with multiple-character regular expressions to multiply the occurrence of a pattern. You can also use parentheses around a single- or multiple-character pattern to remember a pattern for use elsewhere in the regular expression.

To create a regular expression that recalls a previous pattern, you use parentheses to indicate a remembered specific pattern and a backslash (\) followed by an integer to reuse the remembered pattern. The integer specifies the occurrence of the parentheses in the regular expression pattern. If you have more than one remembered pattern in your regular expression, then \1 indicates the first remembered pattern, \2 indicates the second remembered pattern, and so on.

This regular expression uses parentheses for recall:

a(.)bc(.)\1\2

This regular expression matches an a followed by any character (call it character 1), followed by bc, followed by any character (character 2), followed by character 1 again, and then followed by character 2 again. The regular expression can match aZbcTZT. The software remembers that character 1 is Z and character 2 is T and then uses Z and T again later in the regular expression.