APPENDIX A

Understanding Regular Expressions, Special Characters, and Patterns

This appendix describes regular expressions, special or wildcard characters, and patterns used with filters to search through command output. Filter commands are described in the "Filtering show Command Output" section on page 5-9.

Contents

Regular Expressions

A regular expression is a pattern (a phrase, number, or more complex pattern).

- Regular expressions are case sensitive and allow for complex matching requirements. Simple regular expressions include entries like Serial, misses, or 138.
- Complex regular expressions include entries like 00210... , (is), or [Oo]utput.

A regular expression can be a single-character pattern or multiple-character pattern. It 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. The pattern in the command output is called a string.

The simplest regular expression is a single character that matches the same single character in the command output. Letter (A–Z and a–z), digits (0–9), and other keyboard characters (such as ! or ~) can be used as a single-character pattern.

Special Characters

Certain keyboard characters have special meaning when used in regular expressions. Table A-1 lists the keyboard characters that have special meaning.

Table A-1 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.
^	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 use these special characters as single-character patterns, remove the special meaning by preceding each character with a backslash (\). In the following examples, single-character patterns matching a dollar sign, an underscore, and a plus sign, respectively, are shown.

\$ _ \+

Character Pattern Ranges

A range of single-character patterns can be used to match command output. To specify a range of single-character patterns, enclose the single-character patterns in square brackets ([ ]). Only one of these characters must exist in the string for pattern-matching to succeed. 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 lowercase or uppercase alphabet.

Simplify a range of characters by entering only the endpoints of the range separated by a dash (–), as in the following example:

[a–dA–D]

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

[a–dA–D\–]

A bracket (]) can also be included as a single-character pattern in the range:

[a–dA–D\–\]]

Invert the matching of the range by including a caret (^) at the start of the range. The following example matches any letter except the ones listed:

[^a–dqsv]

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

[^\]d]

Multiple-Character Patterns

Multiple-character regular expressions can be formed by joining letters, digits, and keyboard characters that do not have a special meaning. With multiple-character patterns, order is important. The regular expression **a4%** matches the character **a** followed by a **4** followed by a **%**. If the string does not have a4%, in that order, pattern matching fails.

The 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!, and a2 are all valid matches for the regular expression.

Put a backslash before the keyboard characters that have special meaning to indicate that the character should be interpreted literally. Remove the special meaning of the period character by putting a backslash in front of it. For example, when the expression **a\.** is used in the command syntax, only the string a. is matched.

A multiple-character regular expression containing all letters, all digits, all keyboard characters, or a combination of letters, digits, and other keyboard characters is a valid regular expression. For example: **telebit 3107 v32bis**.

Complex Regular Expressions Using Multipliers

Multipliers can be used to create more complex regular expressions that instruct Cisco IOS XR software to match multiple occurrences of a specified regular expression. Table A-2 lists the special characters that specify “multiples” of a regular expression.

Table A-2 Special Characters Used as Multipliers

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

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

a*

The following pattern requires that at least one occurrence of the letter a in the string be matched:

a+

The following pattern matches the string bb or bab:

ba?b

The following string matches any number of asterisks (*):

******

To use multipliers with multiple-character patterns, 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, the following pattern matches one or more instances of alphanumeric pairs:

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

The order for matches using multipliers (*, +, and ?) 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. Thus, the regular expression matches A9b3, but not 9Ab3 because the letters are specified before the numbers.

Pattern Alternation

Alternation can be used to specify alternative patterns to match against a string. Separate the alternative patterns with a vertical bar (|). Only 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.

Anchor Characters

Anchoring can be used to match a regular expression pattern against the beginning or end of the string. Regular expressions can be anchored to a portion of the string using the special characters shown in Table A-3.

Table A-3 Special Characters Used for Anchoring

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

For example, the regular expression **^con** matches any string that starts with con, and **sole$** matches any string that ends with sole.

In addition to indicating the beginning of a string, the ^ can be used to indicate the logical function “not” when used in a bracketed range. For example, the expression **[^abcd]** indicates a range that matches any single letter, as long as it is not the letters a, b, c, and d.

Underscore Wildcard

Use the underscore to match the beginning of a string (^), the end of a string ($), parentheses (( )) , space ( ), braces ({ }), comma (,), and underscore (_). The underscore can be used to specify that a pattern exists anywhere in the string. For example, **_1300_** matches any string that has 1300 somewhere in the string and is preceded by or followed by a space, brace, comma, or underscore. Although **_1300_** matches the regular expression {1300_, it does not match the regular expressions 21300 and 13000t.

The underscore can replace long regular expression lists. For example, instead of specifying **^1300( ) ( )1300$ {1300, ,1300, {1300} ,1300, (1300**, simply specify **_1300_**.