APPENDIX F

Regular Expressions

Several Apache directives allow the use of regular expressions for pattern matching, for example, the `<DirectoryMatch>` container, the `AliasMatch` directive, and of course `mod_rewrite`'s `RewriteRule` directive. This appendix provides a quick guide to the style of regular expressions used by Apache.

Search Patterns

The basic role of a regular expression engine is to take a search pattern and see if a string that matches the pattern occurs in its input: If it does, the match succeeds; if it doesn't, the match fails. At its most basic, this allows you to see if a particular sequence of characters occurs in a line. For example, the regular expression `apache` will match any line that contains the sequence of characters *apache* at any point.

You can use the period (`.`) as a wildcard character for a placeholder for any character. Alternatively, you can construct a wildcard that'll match a specific set of characters using a square-bracket construction called a *character class* (see Table F-1).

Table F-1. Wildcards

Pattern	Description
`.`	Matches any character
`[aeiou]`	Matches any lowercase vowel
`[aeiouAEIOU]`	Matches any vowel
`[a-z]`	Matches any lowercase letter
`[a-zA-Z]`	Matches any letter
`[0-9]`	Matches any digit

You can list all of the characters you want to match, or you can use ranges (such as `a-z`) to signify a continuous range of characters. To specify the minus sign itself, place it at the end of the list. Sensibly, a period in a character class represents a literal period, not a match for any character.

You can also invert these sets to match any character that isn't listed (see Table F-2).

Table F-2. Ranges

Pattern	Description
`[^a-zA-Z]`	Matches anything except a letter
`[^0-9]`	Matches anything but a digit
`[^a-zA-Z0-9]`	Matches any nonalphanumeric character

This allows you to use a regular expression such as `[Aa]pache` to match both *Apache* and *apache*.

There are also two special characters, caret and dollar, that match the start or end of the line, allowing you to search for lines that start or end with a matching string or for entire lines that match. These are known as *anchors* because they anchor the regular expression to the start or end of the text to be matched (see Table F-3).

Table F-3. Anchors

Option	Description
`^`	Anchor regular expression at start
`$`	Anchor regular expression at end

Table F-4 contains some examples.

Table F-4. Wildcard Examples

Example	Description
`^apache`	Matches any line that starts with *apache*
`apache$`	Matches any line that ends with *apache*
`^apache$`	Matches any line that consists of just the word *apache*

Regular expressions also allow for multipliers (see Table F-5); that is, the characters that modify the behavior of the previous matching character to allow it to match multiple characters.

Table F-5. Multipliers

Multiplier	Description
`?`	Matches either zero or one instances of the character
`+`	Matches one or more instances of the character
`*`	Matches zero or more instances of the character

Table F-6 contains some examples.

Table F-6. Multiplier Examples

Example	Description
`hello?`	Matches *hell* or *hello*
`hello+`	Matches *hello*, *helloo*, *hellooo*, *helloooo*, and so on
`hello*`	Matches *hell*, *hello*, *helloo*, *hellooo*, *helloooo*, and so on

These multipliers are particularly powerful when combined with wildcards (see Table F-7).

Table F-7. Wildcards and Multipliers

Pattern	Description
`[01]*`	Matches any binary number (for example 0010, 1001101110001, and so on).
`[a-zA-Z]+`	Matches any word consisting of at least one letter.
`.*`	Matches any sequence of characters.
`.+@.+`	Matches an e-mail address. You may want to use a more specific wildcard than `.` to ensure that all the characters are valid for an e-mail address.

If you want to use any characters that have special meanings literally, you must escape them using a backslash (`\`); see Table F-8 for more information.

Table F-8. Escapes

Escape	Description
`\?`	Matches a ? character
`\+`	Matches a + character
`\*`	Matches an * character
`\.`	Matches a `.` character
`\[`	Matches a [character
`\\`	Matches a \ character

Pattern Extraction and Replacement

By including parentheses (round brackets) in the search pattern, you can extract part of the matched string and use it in a replacement pattern. To illustrate this, the following is a regular expression you might use in Apache to match a URL for a GIF file:

```
^/images/[a-zA-Z0-9]+\.gif$
```

What if you want to use the name of the file as a parameter for a CGI script that dynamically generates the image? To extract parts of the filename, rewrite the search pattern as follows:

```
^/images/([a-zA-Z0-9]+)\.(gif|jpg|png)$
```

Now you can access the name and extension of the file in your replacement pattern using the notation `$n`, where `n` is a number from `1` to `9`, referring to the contents of the *n*th extracted (bracketed) pattern from the left of the search pattern. Here `$1` contains the basename of the file, and `$2` contains the file extension. Note that parentheses are also used for grouping alternatives as the second pair illustrates, but you're not obliged to use an extracted value if the intent of the parentheses is to group only rather than to extract.

If brackets nest, then they're counted in order of the appearance of the left bracket. Additionally, the whole of the matched text is stored in `$0`. SSI commands may also make use of these values from Apache 2 onward.

Using the previous example, you can now rewrite the URL to feed the extracted values to a CGI script using the following replacement pattern:

```
/cgi-bin/image.cgi?image=$1&format=$2
```

Now a request for the URL `/images/test.png` will be rewritten as a request for this:

```
 /cgi-bin/image.cgi?image=test&format=png
```

Both `AliasMatch` and `RewriteRule` can be used for this task. For more information, see Chapter 5.