Appendix A
Pattern Matching

This appendix describes the pattern-matching scheme used by the X.25 switching feature.

Regular Expression Pattern Matching

The X.121 address and Call User Data are used to find a matching routing table entry. The list is scanned from the beginning to the end and each entry is pattern-matched with the incoming X.121 address and Call User Data to the X.121 and Call User Data in the routing table entry. If the pattern match for both entries succeeds, then that route is used. If the incoming call does not have any Call User Data, then only the X.121 address pattern match need succeed with an entry which only contains an X.121 pattern. If Call User Data is present, and while scanning a route is found that matches the X.121 address, but the route does not have a Call User Data pattern, then that route is used when an exact match cannot be found.

Regular expressions are used to allow pattern-matching operations on the X.121 addresses and Call User Data. The most common operation is to do prefix matching on the X.121 DNIC field and route accordingly. For example, the pattern ^3306 will match all X.121 addresses with a DNIC of 3306. The caret (^) is a special regular expression character that says to anchor the match at the beginning of the pattern.

If a matching route is found, the incoming call is forwarded to the *next hop* depending on the routing entry. If no match is found, the call is cleared. If the route specifies a serial interface running X.25, the call will attempt to be forwarded over that interface. If the interface is not operational or out of available virtual circuits, the call will be cleared. Otherwise, the expected Clear Request or Call Accepted message will be forwarded back towards the originator. The "null 0" interface can be used to early-terminate or refuse calls to specific locations.

If the matching route specifies an IP address, a TCP connection will be established to port 1998 at the specified IP address which must be another Cisco router. The Call Request packet will be forwarded to the remote router where it will be processed in a similar fashion. If a routing table entry is not present or the serial interface is down or out of virtual circuits, a Clear Request will be sent back and the TCP connection will be closed. Otherwise, the call will be forwarded over the serial interface and the expected Clear Request or Call Accepted packet will be returned. Incoming calls received via TCP connections that match a routing entry specifying an IP address will be cleared. This restriction prevents Cisco

routers from establishing a TCP connection to another router which would establish yet another TCP connection. A router must always connect to the remote router with the destination DTE directly attached.

Regular expressions provide a way to specify wide ranges of X.121 addresses and Call User Data fields by using just a few keystrokes. If you are familiar with regular expressions from UNIX programs such as *regexp*, you are already familiar with much of Cisco Systems' regular expression implementation.

Writing regular expressions is simple once you see and try a few examples. A regular expression is a formula for generating a set of strings. If a particular string can be generated by a given regular expression, then that string and regular expression *match*. In many ways, a regular expression is a program, and the regular expression matches the strings it generates.

A regular expression is built up of different components, each of which is used to build the regular expression string-generating program. The simplest usable component is the *atom*, but first you need to understand *ranges*, as atoms are built of these.

Ranges

A range is a sequence of characters contained within left and right square brackets ([ ]). A character matches a range if that character is contained within the range; for example, the following syntax forms the range consisting of the characters "a," "q," "c," "s," "b," "v," and "d." The order of characters is usually not important; however, there are exceptions and these will be noted.

[aqcsbvd]

You can specify an ASCII sequence of characters by specifying the first and last characters in that sequence, and separating them with a hyphen (-).

[a-dqsv]

The above example could also be written so as to specify right square brackets ( ]) as a character in a range. To do so, enter the bracket as the *first* character after the initial left square bracket that starts the range.

This example matches right bracket and the letter "d."

[ ]d]

To include a hyphen (-), enter it as either the first or the last character of the range.

You can reverse the matching of the range by including a wedge (caret) at the start of the range. This example matches any letter *except* the ones listed. When using the wedge with the special rules for including a bracket or hyphen, make the wedge the very first character.

   [^a-dqsv]

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

   [^]d

Atoms

Atoms are the most primitive usable part of regular expressions. An atom can be as simple as a single character. The letter "a" is an atom, for example. It is also a very simple regular expression, that is, a program that generates only one string, which is the single-letter string made up of the letter "a." While this may seem trivial, it is important to understand the set of strings that your regular expression program generates. As will be seen in upcoming explanations and examples, much larger sets of strings can be generated from more complex regular expressions.

Certain characters have a special meaning when used as atoms; refer to Table A-1.

Table A-1   Special Symbols Used as Atoms

.	Matches any single character
^	Matches the null string at the beginning of the input string
$	Matches the null string at the end of the input string
\ **character**	Matches *character*

S-1639

Note another use for the ^ symbol.

As an example, the regular expression matches "abcd" only if "abcd" starts the full string to be matched:

   ^abcd

Whereas the following expression is an atom that is a range that matches any single letter, as long as it is not the letters "a," "b," "c," or "d."

   [^abcd]

It was previously stated that a single character string such as the letter "a" is an atom. A character by itself, such as **$**, means "match the null string at the end of the input string."

> **$**

Whereas this atom matches a dollar sign ($). Preceding a character with a backslash (\) removes the special meaning of that character.

> **\$**

Any character can be preceded with the backslash character with no adverse affect.

> **\a**

Atoms are also full regular expressions surrounded by parentheses. For example, both "a" and "(a)" are atoms matching the letter "a." This will be important later, as we see patterns to manipulate entire regular expressions.

Pieces

A piece is an atom optionally followed by one of the symbols listed in Table A-2:

Table A-2 Special Symbols Used with Pieces

*	Matches 0 or more sequences of the atom
+	Matches 1 or more sequences of the atom
?	Matches the atom or the null string

S1640

For example, matches any number of occurrences of the letter "a," including *none.*

> **a***

This string requires there to be at least one letter "a" in the string to be matched:

> **a+**

This string means that the letter "a" can be there *once*, but it does not have to be:

> **a?**

This string matches any number of asterisks (*).

> *****

Here is an example using parentheses. This string matches any number of the two-atom string "ab."

(ab)*

As a more complex example, this string matches one or more instances of letter-digit pairs (but not none, that is, an *empty string* is not a match):

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

The order for matches using the optional *, +, or ? symbols is longest construct first. Nested constructs are matched from outside to inside. Concatenated constructs are matched beginning at the left side of the construct.

Branch

A branch is simply a set of zero or more concatenated pieces. The previous letter-digit example was an example of a branch as concatenated pieces. Branches are matched in the order normally read—from left to right. For example, in the previous example, the regular expression matches "A9b3," but not "9Ab3" because the alphabet is given first in the two-atom branch of [A-Za-z][0-9].

Regular Expressions

A regular expression is a branch or any number of branches separated by a vertical bar (|). A string is said to match the regular expression if it is generated by the "program" specified in any of the branches. Of course, a string can be generated by more than one branch. For example, "abc" is generated by all branches in the regular expression

abc|a*(bc)+|(ab)?c|.*

Also remember that if a regular expression can match two different parts of an input string, it will match the earliest part first.

The regular expression support and the technical information for this portion of the documentation is based on Henry Spencer's public domain *rgrep(3)* library package.