Published: 01/26/67

Identification

Searching Rule Statements
S.L. Rosenbaum

Purpose

When a user wishes to provide a special searching technique for finding segments in the file system hierarchy, he takes the following steps:

1. constructs searching rule statements in the language recognized by the Search Module

2. stores searching rule statements in segments

3. tells the Search Module which segments to use for searching (sets the system option "search")

This paper describes the syntax and vocabulary of the language for constructing searching rules.

Usage

A searching rule statement consists of one of the keywords which this paper defines followed by any relevant arguments. One or more blanks separate the keyword from its arguments and the arguments from each other; a semi-colon separates consecutive searching rule statements, i.e.,

```
     keyword1 arg11 arg12; keyword2;...
```

The user can insert extra horizontal and vertical spacing characters to increase readability. Parentheses must enclose an argument which is a list of more than one element. The user can label any searching rule statement by preceding its keyword with a character string followed by a colon. The character string is the label of the statement, i.e.,

```
     label1:keyword1 arg11 arg12; label2:keyword2;...
```

Special characters

The Search Module attaches a special meaning to some graphic characters in specific contexts. This special interpretation is built into the Search Module's scanning mechanism and in general follows the conventions established by section

BX.8.00--"Overview of File System Commands", and the conventions of section BX.1.00--"The Multics Command Language".

Throughout this section a special interpretation of a character is described when contextually meaningful. A summary of the special interpretations appears in Appendix II at the end of this section.

The "special" characters used in common with the file system commands are:

1. the asterisk-- *

2. the period-- .

3. the "greater than"-- >

4. the "less than"-- <

In common with the Shell, the Search Module uses:

1. the horizontal space-- <sp>

2. the semi-colon-- ;

3. the colon-- :

4. pairs of parentheses-- (and)

5. pairs of braces-- {and}

6. the per cent sign-- %

7. the new line-- <NL>

Changing the data base of special characters (section BX.2.01) affects both the Shell and the Search Module.

In general, the Search Module processes searching rule statements sequentially, ignoring user inserted comments. A comment is a statement beginning with the keyword "c", i.e.,

```
     c This is a comment;
```

<u>Processing failures</u>

A processing failure results when the Search Module cannot successfully process a searching rule statement. Some causes for processing failure are:

1. an unrecognizable keyword

2. the wrong number of arguments for a keyword

3. the inability to perform the operations indicated by a keyword

The discussion for each keyword includes a description of the specific circumstances which can cause the keyword's processing failure. The procedure followed in the event of a processing failure depends upon the specific processing failure and the status of the system options "brief" and "go_ahead". (See BX.12.00 for a discussion of options.) When a processing failure occurs, the Search Module terminates the processing of the searching rule statement at fault. It puts messages in the user's output stream and in his error segment, errout, which describe the reason for termination. If the "brief" option is off, a detailed message is put in the output stream; if "brief" is on, an abbreviated form of the message is used. (Appendix III at the end of this section lists the error codes and their related messages which can be sent to the user.) The Search Module then checks the setting of the "go_ahead" option to see if it should return to its caller or resume processing. When the "go_ahead" option is "off" the Search Module signals to its caller with an error code which indicates the reason for the processing failure. If the "go_ahead" option is "on", the Search Module resumes processing with the next statement after the one at fault.

Escape character

The Search Module's escape character normally is the per cent sign "%". When the user wants to use a graphic character in a context which ordinarily has a special meaning to the Search Module, he precedes the character in question with the escape character. For example, he wants to use the colon, which normally terminates a label, as part of a label. For example,

```
label1%::keyword1;
```

defines the character string "label1:" as the label of the searching rule statement.

Path names

Some keywords take path names as arguments. (See section BX.8.00 for a discussion of path names.) The Search Module interprets path name arguments according to the following rules:

1. {root} represents the root directory.

2. an absolute path name is a path name relative to the root directory and begins with the character ">" or one of the special character strings enclosed by braces which represents an absolute path name (see 3., 4., 5., and 6.).

3. {wdir} represents the absolute path name of the current working directory.

4. {pdir} represents the absolute path name of the current process directory.

5. {ldir} represents the absolute path name of the current system library directory.

6. {cdir} represents the absolute path name of the calling directory, i.e., the directory of the procedure which wants the segment being sought.

7. all other path names are interpreted relative to the current working directory.

For example,

    a > b

represents the path name "a > b" relative to the current working directory.

It can also be represented by:

    {wdir} > a > b

direct, getmatch

When the user wants to specify a particular sequence of directories for the Search Module to search, he uses the keyword "direct". This keyword takes two arguments. The first is an ordered list of directory path names to search; the second is a list of searching rule statements to process if, and only if, the name is found in a directory. Thus,

```
     direct directlist actionlist;
```

means

> "Search directories in *directlist* for the name sought, and only when it is found, do *actionlist*."

For example, the searching rule statement:

```
     direct ( >a >b >c) getmatch;
```

searches directories >a, >b, and >c, in that order, for the name sought.

The parentheses indicate that ">a", ">b" and ">c" comprise the list of elements for the *directlist* argument where elements are separated from one another by blanks. In the above example, parentheses enclosing the *actionlist* argument are optional since the list consists of only one element, "getmatch".

When a matching name is found in one of the directories searched, "getmatch" directs the Search Module to return to its caller with a path name comprised of the path name of the directory containing the matching name concatenated with the matching name. For example, if the Search Module seeks and finds the name "alpha" in directory ">a", it returns the path name ">a> alpha".

The user must specify the directories to search; the Search Module assumes "getmatch" as the default action if the user does not specify any statements to process when a match is found. Notice that the Search Module interprets the statement

```
     direct getmatch;
```

as a request to search the directory relative to the current working directory by the name of "getmatch", i.e., directory "{wdir} > getmatch".

Although the search specified by "direct" may not yield a matching name, only an incorrect form for the *directlist* argument can cause "directs"'s processing failure. A failure exists if:

1. an element of <u>directlist</u> does not represent the path name of a directory.

2. an element of <u>directlist</u> does represent a directory path name but the user does not have access to read the directory.

In the event of a processing failure, the Search Module leaves a message both in the user's error file and in his output stream. If the "go_ahead" option is "off" the Search Module signals its caller with the error code signaling that a processing failure occured. If the "go_ahead" option is "on", the Search Module goes to the next sequential searching rule statement following the "direct" statement at fault. A processing failure occurs for "getmatch" if no matching name is found by the time the Search Module processes "getmatch".

<u>name, name_not</u>

When the user wants to specify restrictions to the Search Module with respect to the name being sought, he can use the "name" and "name_not" keywords. These keywords require two arguments; the first is a list of symbolic entry names with which to compare the name being sought (see section BX.8.00 for a discussion of entry names); the second is a list of searching rule statements. Thus,

name <u>namelist</u> <u>actionlist</u>;

means

"Do <u>actionlist</u> only if the name sought is in <u>namelist</u>; otherwise go to the next searching rule statement."

and

name_not <u>namelist</u> <u>actionlist</u>;

means

"Do <u>actionlist</u> only if the name sought is not in <u>namelist</u>; otherwise, go to the next searching rule statement."

"direct (>a >b >c)" can be written as:

```
name **
     direct (>a >b >c);
```

where "**" means "any name". (See section BX.8.00 for a complete discussion on the use of "*" 's in entry names.)

The searching rule statement

```
     name (*.pl *.epl)

          direct >a>pl;
```

searches the directory ">a>pl" only if the secondary component of the name being sought is either "pl" or "epl".

(Note: The conventions of section BX.8.00 define "*" as "a single component of an entry name" and "**" as "any number of components". Hence, "*.X" means a two component name whose second component is "X"; "**.X" means the last component of any name is "X". The "." character separates the graphic components of an entry name.)

"name" and "name_not" have no special processing failures. The searching rule statement

```
     name_not (*.pl *.epl)

          direct >a>pl.not;
```

searches the directory ">a>pl.not" only if the secondary component of the name being sought is neither "pl" nor "epl".

The following example illustrates how restrictions may be nested.

```
     name (x.* y.* z.*)

          (name *.pl direct >a>pl;

          name *.epl direct >a>epl);
```

The single statement above could also be expressed as the pair of statements:

```
     name (x.pl y.pl z.pl)

          direct >a>pl;

     name (x.epl y.epl z.epl)

          direct >a>epl;
```

caller, caller_not, ring, ring_not

At times the user may want a statement's processing to depend upon the caller of the name for which a segment is sought. He can establish this type of restriction with the "caller", "caller_not", "ring" and "ring_not" keywords. The first pair refer to the name of the caller and the second pair refer to the ring number of the caller.

These keyword statements take the same form as the "name" and "name_not" statements.

caller *callerlist* *actionlist*;

takes the meaning:

"Do *actionlist* if the name sought is wanted by a procedure whose name is in *callerlist*."

caller_not *callerlist* *actionlist*;

means

"Do *actionlist* only if the name sought is wanted by a procedure whose name is not in *callerlist*."

ring *ringlist* *actionlist*;

means

"Do *actionlist* only if the name sought is wanted by a procedure which resides in a ring whose number is in *ringlist*."

and

ring_not *ringlist* *actionlist*:

means

"Do *actionlist* only if the name sought is wanted by a procedure which resides in a ring whose number is not in *ringlist*."

For example:

caller x direct {cdir};

indicates that a procedure named "x" wants its procedures from the same directory in which "x" resides. (Reminder:

{cdir} means the user's current calling directory.) The statement:

```
     ring_not (3 4 5)

          direct {ldir};
```

searches the user's current system library if the procedure seeking the name is not in rings 3, 4, or 5.

Neither "caller" nor "caller_not" have processing failures other than insufficient arguments. The processing failure for "ring" and "ring_not" is an unrecognizable ring number.

<u>getseg</u>

If the user knows the location, i.e., path name, of the segment he wants to use, he can specify it directly with the "getseg" keyword. For example, the searching rule statement:

```
     name sort.* getseg >sort.radix;
```

uses the segment ">sort.radix" if the primary component of the name sought is "sort". The Search Module returns to its caller with the path name ">sort.radix". "getseg" requires as its argument the path name of the segment to be used. The omission of this argument is "getseg"'s processing failure.

<u>return</u>

When the user decides that further searching is in vain, he says:

```
     return;
```

The "return" statement causes the Search Module to return to its caller without a path name, i.e., with a null path name.

<u>go</u>

When the user wants to interrupt sequential processing by transferring to a searching rule statement, he uses the "go" statement. It has the general form:

     go <u>label</u> <u>pathname</u>;

and means

     "Transfer search control to a statement labelled <u>label</u> residing in the segment with the path name <u>pathname</u>."

Upon encountering a "go" statement, the Search Module gets the segment located by pathname and, starting at the beginning of that segment, looks for the first statement labelled label.

For example, the statement:

```
     name sort.*

     go nogot >b;
```

transfers to the first statement labelled "nogot" residing in the segment with the path name ">b" when the primary name sought is "sort". A processing failure occurs if either the Search Module cannot get or cannot read the segment indicated by the path name. In the above example, a processing failure occurs if: 1) a segment with the absolute path name ">b" does not exist, 2) it does exist but the user does not have read access to it or 3) the Search Module cannot find any statement in the segment which is labelled "nogot". If the user omits the path name, the Search Module assumes that the labelled statement resides in the segment currently being processed. In addition, the omission of the path name indicates that the labelled statement appears in the segment after the "go" statement, i.e., the Search Module looks for the first statement labelled label after the "go" statement which resides in the same segment as the "go" statement.

The user must be very careful to avoid creating infinite loops with the "go" statement. Assume that a certain segment starts with the following sequence

```
     statement1:  go statement1;

     statement1:  ...
```

The omission of a path name in the "go" statement causes the Search Module to begin looking for the statement labelled "statement1" after the "go" statement and, hence, the Search Module ignores the fact that the "go" statement itself has the label desired. On the other hand, assume a certain pathological segment has the path name ">path1" and starts with the sequence:

```
     statement1:  go statement1 >path1;

     statement1:  ...;
```

In this case the "go" statement causes the Search Module to start looking for the labelled statement at the beginning of the segment ">path1". The result is an infinite loop around the first statement of the segment ">path1".

In general, it is advisable to keep all labels unique within a single segment.

<u>default</u>

The user can switch to the standard searching rule statements which would have been processed if the <u>search</u> option had been "off". The statement:

```
     default;
```

causes the Search Module to terminate its current processing, obtain the standard searching rule statements and process them. See section BD.4.01 for a description of the standard searching rule statements. No processing failure exists for the "default" statement.

<u>rename</u>

The user can direct the Search Module to look for a name different from the name for which the Seach Module was called. The statement:

```
     rename dummy
```

directs the Search Module to search for a segment named "dummy" instead of the segment name for which it had been searching. The statement:

```
     name (sort.* graph.*)

          rename dummy.*;
```

directs the Search Module to search for a segment with the name with the primary component "dummy" whenever it had been seeking a segment for the name with the primary component "sort" or "graph".

A "rename" is in effect until the Search Module encounters another "rename" and only operates on the name the Search Module is seeking at the current time. For example, in the sequence:

```
     name (sort.* graph.*)

          rename dummy.*;

     name sort.epl default;
```

the second "name" statement is superfluous and its actionlist never gets processed--if the Search Module originally had been looking for a segment named "sort.epl", by the time the Search Module processes the second "name" statement it is seeking a segment named "dummy.epl".

listen, endlisten

If the user wants to give the Search Module some instructions during its actual operation rather than presetting all his directions, he uses the keyword "listen". This statement puts the user in the Search Module's listening mode, an operational mode from which the user can submit searching rule statements for immediate processing. When the Search Module encounters

```
     listen;
```

it prints out a message informing the user that he should submit a statement for the Search Module to process. For example, the sequence:

```
     name sort.* (listen; default);

     direct >a;
```

puts the user into the listening mode when the primary name sought is "sort". After being notified that he is in the listening mode, the user can proceed to obtain information (by calling Search Module procedures as explained later in this section) concerning a processing failure, perform a search, create a suitable segment, try a searching rule statement, etc.. The user inputs a statement or sequence of statements, terminated by the break character (normally the <NL> character). After recognizing the break character, the Search Module performs the indicated actions, notifying the user if an action cannot be processed or completed and returns to the listening mode for further instructions upon processing competion or failure. A successfully executed "go", "getseg", "getmatch", "default" or "return" statement automatically removes the user from the listening mode, and, except for the "go" and "default" statement, returns the user to the Search Module's caller. If the user wishes to return to his pre-set searching rule statements, he types

```
     endlisten;
```

and the Search Module resumes processing with the keyword statement immediately following the "listen" statement. (In the previous example, the Search Module resumes processing with "default", switching to the default searching rule statements.)

When the user has declared himself to be incommunicade by setting the "no_questions" option "on" (see BX.12.00-"Options"), the Search Module treats the "listen" searching rule statement as an error, i.e., the inability of the Search Module to interact with the user is "listen" 's processing failure. (Note: "Endlisten" has no meaning outside the listening mode.)

In the above example, if both the "no_questions" option and the "go_ahead" option are "on", the Search Module processes the "direct" statement.

<u>call</u>

When the user wants to execute a system routine or private procedure, he uses the keyword "call". "call" takes as its argument a character string which is enclosed by a quoting character and which is to be passed to the Shell for interpretation as a command sequence. For example, assume there is a procedure named "alpha" which takes two arguments: a directory path name and an integer. The searching rule statement:

```
     call /alpha >a>b 4/;
```

invokes the procedure "alpha" with ">a>b" and "4" as arguments where the slash character functions as the quoting character. (Any character not occurring within the character string may be used as the quoting character.)

If he wants to use the path name of the directory last searched as an argument instead of ">a>b", he uses the value of the Search Module procedure "smdirect" and says:

```
     call /alpha {smdirect} 4/;
```

The other Search Module procedure values are:

{smname}	- the symbolic name for which the Search Module was called to search
{smrename}	- the symbolic name for which the Search Module is currently searching
{smmatch}	- the directory path name in which the name sought was last found by searching. {smmatch} is a null character string, if no match has been found.
{smlabel}	- the label attached to the last processed labelled statement.

When the Shell encounters the above procedure values, it calls the respective Search Module procedure as an immediate-value command (section BX.1.00).

error

If the user has set the "go_ahead" option "on" and wants to supply an error procedure for a statement which may fail, e.g., the labelled statement for a "go" does not exist, he uses the "error" statement which takes the form:

```
     error actionlist;
```

where actionlist is the list of statements processed only if the previous statement resulted in a processing failure. For example,

```
     direct >a go x;

     error return;
```

directs the Search Module to return to its caller if ">a" is not a path name of a directory to which the user has read access. In this example, "error" applies to the "direct" statement and not the "go". On the other hand, the sequence:

```
     direct >a (go x;error return);
```

directs the Search Module to return to its caller if the "go" statement fails, i.e., no statement labelled "x" appears in the same file as and after the "go" statement.

APPENDIX I - Keywords

keyword	arguments	processing failure for keyword
call	commandlist	error return from Shell
caller	callerlist actionlist	none
caller_not	callerlist actionlist	none
default	none	none
direct	directlist actionlist	directlist incorrect
endlisten	none	not in "listen" mode
error	actionlist	none
getmatch	none	no matching name found
getseg	pathname	no accessible segment for pathname
go	label pathname	no accessible statement for label
listen	none	no_questions option is "on"
name	namelist actionlist	none
name_not	namelist actionlist	none
rename	name	none
return	none	none
ring	ringlist actionlist	ringlist incorrect
ring_not	ringlist actionlist	ringlist incorrect

APPENDIX II - Special Character Set

graphic	interpretation	comments
<sp>	separates arguments	
;	separates searching rule statements	need not be followed by a <sp>
:	ends a statement label	"
(and)	begins and ends a list, respectively	
<NL>	ends a request	only in "listen" mode
%	escape character	
{"char"}	value of function "char"	only in path names or sent to the Shell
*	one component of a segment name	only where segment name allowed
**	any number of components of a segment name	"
.	separates name components	"
<	one level superior	only where path names allowed
>	one level inferior	"

Appendix III to be specified later