To: OS Group
OS Interest List

From: Alan Dossett (userid: ALAN, x3972)

Date: 23 October 1984

Subject: Functional Specification of Search Rules in Primos

Attached is the preliminary functional specification for search rules in Primos. Please return any comments to me by Friday 31 October 1984.
# Table of Contents

1 Functions
   1.1 Terminology ...................................................... 1
   1.2 Goals and Non-Goals ............................................. 2
   1.3 The User's Model .................................................. 3
   1.4 Product Functions ................................................. 4
      1.4.1 Search Lists .................................................. 4
      1.4.2 Set_Search_Rules (SSR) Command ............................ 11
      1.4.3 List_Search_Rules (LSR) Command ............................ 15
      1.4.4 Expand (EXP) Command/Function ............................. 17
      1.4.5 Attach-Scan .................................................... 20
      1.4.6 Command Search ............................................... 22
      1.4.7 New Routines .................................................. 24
      1.4.8 OPN$SR - Open an Object Using Search Rules ............. 25
      1.4.9 OPN$SRSF - Open an Object Using Search Rules and Suffixes ........................................................................... 29
      1.4.10 SR$CREAT - Create a Search List ........................... 32
      1.4.11 SR$DEL - Delete a search list ............................... 34
      1.4.12 SR$INIT - Initialize a Users Search Lists ............... 35
      1.4.13 SR$ADDB - Add Rule Before a Given Rule ............... 37
      1.4.14 SR$ADDE - Add Rule to End (After Given Rule) ........ 39
      1.4.15 SR$REM - Remove a Rule ....................................... 41
      1.4.16 SR$TEMP - Process a Template File ....................... 43
      1.4.17 SR$NEXTR - Get Next Rule ................................. 45
      1.4.18 SR$SETL - Set Locator ......................................... 48
      1.4.19 SR$LIST - Get Names of Defined Lists ................... 50
      1.4.20 SR$READ - Read the Rules in a List ..................... 52
      1.4.21 SR$FR_LS - Free List Structures ........................... 54

2 Testability ........................................................................ 54

3 Design Notes .................................................................... 55

4 Reliability ...................................................................... 55

5 Performance .................................................................... 55

6 Configuration ................................................................... 55

7 Installation ..................................................................... 55

8 Ease of Use ..................................................................... 56

9 Maintainability ............................................................. 56

12 October 1984 Functional Specification
Search Rules in Primos

10 Compatibility ................................................................. 56
11 Standards ................................................................. 56
12 References ................................................................. 56
Search Rules in Primos

1 Functions

This document is the functional specification for the project to add general search rules to Primos. Today, Primos uses search rules for dynamic linking to library entrypoints. This specification details search lists for attach-scan, commands, and general program use.

A search rule is like a clue for finding an object. If the object is not found using a given rule, another rule is used if possible. A search list originates as a text file that contains rules. Each list has a specific name and is used by one or more mechanisms for which the rules have meaning. For example, ENTRY$.SR is the name of a list of rules to use when dynamically linking to an entrypoint. ENTRY$.SR only has meaning to the dynamic linker; neither the search rules mechanism nor any other mechanism (like a compiler, or the copy command) knows what those rules mean. The search rules mechanism provides a means for the linker to use the rules, but it does not interpret them.

The rules in a list are ordered and used that way. The first rule is used first to find an object. If the object is not found with the first rule, the second is used, and so on until either the object is found or there are no more rules.

There is one major difference between the functions in the proposal and the functions described in this document: the method for using remote partitions will continue to be the same as it is today; ADDISK will be used to add the partition. The proposal said that remote partitions would be specified exclusively through search rules.

This document specifies the functions of some commands and routines that are already part of Primos but that have not been formally specified before. In most cases, no changes have been made to these commands and routines.

1.1 Terminology

administrator rules--search rules kept in SYSTEM>ADMIN$. (COMMAND$ ATTACH$ ENTRY$).SR that are always put at the beginning of the user's in-memory copy of search rules so they are always used first.

attach-scan--the method by which Primos searches partitions to find file system objects.

command--an executable file.

default rules--search rules kept in SYSTEM>(COMMAND$ ATTACH$ ENTRY$).SR to which the user's rules are defaulted.
**Search Rules in Primos**

**search list**—a list of search rules managed by the search rules mechanism.

**search rule**—a partition name (like <OSGRP1>), a pathname (like ALAN>COMMANDS), or an expression that, when evaluated, is one of these two. Used to specify the name of a location to search when looking for an object.

**system rules**—synonomous with default rules.

**template file**—an editable text file that contains the search list.

**user rules**—search rules in any template file that may be different from default or administrator rules and are set by the user.

### 1.2 Goals and Non-Goals

The following are goals of this project:

1. Add search rules for attach-scan to Primos.
2. Add search rules for commands to Primos.
3. Make the search rules mechanism available for any program to use.

The following are **not** goals of this project:

1. Allow information about search rules to be specified on a profile, group, or project basis. Any default or administrator rules are system-wide.
2. Make changes to Primix search rules.
3. Change other mechanisms, e.g., compilers or spooler, to use search rules.
4. Allow search rules on a program level instead of process basis.
5. Increase the number of partitions on the system.
Search Rules in Primos

1.3 The User's Model

Attach-Scan

Attach-scan search rules enable a user to specify an ordered list of partitions, both local and remote. This list is used when Primos searches for file system objects (e.g., directories, and files) that are not fully qualified (i.e., no partition is given). When fully qualified objects are specified, the search rules are not used.

Command Search

Command search rules enable a user to specify an ordered list of directories (they need not be fully qualified). This list is used when Primos searches for a command to execute. The search rules are used when just the command name and not its location is given. (Note that commands are executable files with no suffix or one of the following suffices: .cpl, .save, .run. Files that end with .seg must be executed explicitly with the seg command.)

General Program Use

People may choose to incorporate their own lists of search rules into their own programs. For this reason, the search rules procedures will be documented and made available to users.

Use with Primix

Users of Primix should see no conflict between Primos and Primix search rules: Primos maintains its own search rules as does Primix. When issuing a single Primix command from Primos, the Primix rules will be used for that command. When issuing a single Primos command from Primix, the Primos rules will be used.

Search Rules for the Naive User

The naive user will not know about search rules. Default rules would be in effect and could make Primos look exactly as it does today. Search rules will be used primarily by the advanced user or
writer of applications subsystems.

Command Levels

When a user is at command level n and sets search rules different than those at level n-1, the new search rules will be used for all command levels from 1 to n, not just for level n. This is done to eliminate confusion; the user need not know what level she is at when she sets her search rules. The natural assumption is that changing rules at one level changes them at all. This is analogous to setting one's current attach point.

The System Administrator's Model

The only changes that specifically affect the system administrator are the need to create a default attach-scan search list, and the option to create an administrator list (this is not necessary).

Phantoms and Children

When a phantom is spawned, it will have system default search rules, not the rules of the spawner. Children, however, do have the rules of the parent at the time the child is spawned.

1.4 Product Functions

1.4.1 Search Lists

This section describes the format of the search list template file. This file is made up of rules, with one rule per line. Rules are either partition names (like <0SGRP1>), pathnames (like ALAN>COMMANDS), or expressions that evaluate to one of these two.

List names are used to identify a list. According to the Prime convention, list names with a dollar-sign are reserved for
Prime use, i.e., users should not create list names with dollar-signs in them.

A user's in-memory copy of a search list can be made up of one, two, or three components: user-defined rules, default rules, and administrator rules. The user has control over the inclusion of user-defined rules and of default rules but the system administrator has control over the inclusion of administrator rules. Default and administrator rules are specified on the basis of list name. Further detail is given in later sections.

1.4.1.1 Syntax/Form

Following is the BNF for a search list. The symbols (, ), {, and } are metasymbols; the symbols -, <, >, [, and ] are taken literally.

(search list) ::= {(rule)}

(rule) ::= (partition name)
          | (pathname)
          | (partition name)(pathname)
          | (SR expression)
          | (blank line)

(partition name) ::= <(legal 6-character name)>

(pathname) ::= (legal 32-character name)>(pathname)
          | (legal 32-character name)
Search Rules in Primos

\[(SR \text{ expression}) ::= [\text{home\_dir}] \mid [\text{home\_dir}\rangle(\text{pathname}) \mid [\text{origin\_dir}] \mid [\text{origin\_dir}\rangle(\text{pathname}) \mid [\text{referencing\_dir}] \mid [\text{referencing\_dir}\rangle(\text{pathname}) \mid -\text{use } (\text{partition name})(\text{pathname}) \mid -\text{use } (\text{pathname}) \mid -\text{system} \mid -\text{eval } (\text{CPL expression}) \mid -\text{static\_mode\_libraries} \mid -\text{added\_disks}\]

Notes

The maximum length of a search rule is 128 characters.

Throughout this document, parentheses are used to denote the nonterminals of a grammar instead of angle brackets. This is done to eliminate ambiguity.

All keywords and functions must be given literally, as above; there are no abbreviations.

1.4.1.2 Semantics/ Meaning

Search lists are composed of search rules, one per line. The ordering of the rules determines the order in which they are used. A search rule is either a partition name, a pathname, or an expression that evaluates to one of these. A search rule expression (SR expression in the BNF) is either a (pseudo-CPL) function (like [home_dir]), or a keyword that begins with a ‘-‘.
**Search Rules in Primos**

<table>
<thead>
<tr>
<th>SR expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[home_dir]</td>
<td>current attach point when at command level</td>
</tr>
<tr>
<td>[origin_dir]</td>
<td>initial attach point</td>
</tr>
<tr>
<td>[referencing_dir]</td>
<td>directory from which reference is being made (see example below)</td>
</tr>
<tr>
<td>-use (pathname)</td>
<td>use the search list specified by (pathname) at this point in the search list. The list is evaluated at the time the Set_Search_Rules command is given.</td>
</tr>
<tr>
<td>-system</td>
<td>use the system rules of this list_name at this point in the search list. System (also called default) rules are located at SYSTEM&gt;(list name).SR.</td>
</tr>
<tr>
<td>-eval (CPL expression)</td>
<td>evaluate the CPL expression at the time the Set_Search_Rules command is given. The expression must evaluate to a valid rule.</td>
</tr>
<tr>
<td>-static_mode_libraries</td>
<td>relevant only for the ENTRYS list. This means search the static mode libraries for the entrypoint.</td>
</tr>
<tr>
<td>-added_disks</td>
<td>relevant only for the ATTACHS list. This means search the added disks.</td>
</tr>
</tbody>
</table>

**Example of [referencing_dir]**

The function [referencing_dir] is best described by example. A user runs a program using special search rules that is in a directory other than her current attach point and that program assumes the existence of a file in the same directory. The special search rules are used to find that file.

attach alan

/* set search rules for my_prog and run the program
set_search_rules alan>tools>my_prog.sr -ns
r alan>tools>my_prog  /* assumes the existence of
/* alan>tools>my_prog.input
The program would fail if the my_prog.sr search rules were

<osgrpl>alan
[home_dir]
because my_prog.input is not present in <osgrpl>alan. But the program would work if the rules were

<osgrpl>alan
[home_dir]
[referencing_dir]

because [referencing_dir] evaluates to <osgrpl>alan>tools.

The -added disks Keyword

This rule is evaluated at use-time to a list of the added disks; when searching down the ATTACH$ list and finding the rule -added_disks, the attach-scan will then search all of the added disks on the system in order, one at a time. This list is not included in the in-memory copy at the time of the SSR command so that all disks added at the time of the attach-scan will potentially be searched.

Users are discouraged from using this rule because it can cause duplication of effort. If a user's ATTACH$ list contains several partitions followed by the -added_disks rule, the first partitions searched will be searched again in the course of searching the added disks. The only time the -added_disks rule should be used is either in the default search list (SYSTEM>ATTACH$.SR) or if a user anticipates a constantly changing list of added partitions. In the latter case, the user would want to be sure of searching all partitions as soon as they are added. The -added_disks rule should never be in the administrator rules (see below).

In general, the Operating Systems Group should refrain from adding keywords that do not have generic meaning across all lists. This is to avoid a tangle of some rules being correct for some lists and not for others. (Examples of rules that do not follow this maxim are -added_disks and -static_mode_libraries. A rule that does follow is -system)

Administrator Rules

For compatibility reasons, the system administrator might need to designate rules that are always used first before any default or user rules. These "search-first" rules are always put at the beginning of the user's in-memory search list.

Administrator rules, if they are necessary, are kept in SYSTEM>ADMIN$. (COMMAND$ ATTACH$ ENTRY$).SR. They are
incorporated into a users search list when the user logs in, and whenever the user issues the SSR command.

The administrator ATTACH$ list (SYSTEM>ADMIN$, ATTACH$, SR) should never contain the -added_disks rule. If it did, users would have no flexibility with attach-scan and a performance degradation might be seen because of duplication of effort.

In general, the system administrator should be discouraged from defining administrator rules. Such rules limit flexibility for users, could cause performance problems, and could prevent the user from increasing her performance by making an intelligent ordering of her search rules.

Administrator rules are only necessary for attach-scan if there are programs or applications packages running on the system that must read or write specific but unqualified files. These files would be used, for example, for accounting or audit trails and therefore the correct file must be found independent of the user's search rules. For example, if an accounting routine writes data to ACCOUNTING>USER17, and assumes that ACCOUNTING is on logical device 0, the user could set search rules that would force the attach-scan to find it on some other partition. If the accounting package is not flexible enough to handle that, it might never bill user17.

(Administrator rules should not be used as an excuse for writing new accounting packages as described above because they limit the ability of the user to increase performance through intelligent ordering of attach-scan rules. Rather, new accounting packages should be impervious to a user's search rules by obtaining and using fully qualified pathnames through other means.)

Administrator rules might also be set for the lists COMMAND$ and ENTRY$. This is provided more for symmetry than need. It typically will not be used.

1.4.1.3 Operational Procedures

To use search lists, one creates the template file containing rules, then issues the Set_Search_Rules command.
1.4.1.4 Errors

The following are flagged as errors at the time of the Set_Search_Rules command: duplicate rules, and unrecognized functions.

1.4.1.5 Restrictions

There are no restrictions other than noted in the Errors section above.
1.4.2 Set_Search_Rules (SSR) Command

This command makes the search rules mechanism aware of a particular list name and the rules of that list for the user. The user can think of SSR as "registering" his rules by name with the search rules mechanism.

When SSR is issued, the search list is read from the file (which contains the template) and placed in the user's address space. The template becomes a search list that is then used by the search rules mechanism. Every time the user changes the rules in the file, he should reissue the SSR command so that the search rules mechanism is aware of the change. If he does not, the old rules remain in effect because they are the ones in the user's address space.

1.4.2.1 Syntax/Form

The BNF for the Set_Search_Rules command is given below. Capital letters in literal words indicate what part of the literal must be given.

(SSR command) ::= Set_Search_Rules (partition name)(pathname) (optional args) |
| Set_Search_Rules -DeFauLT -List_NAME (list name) |
| Set_Search_Rules -DeFauLT (list name) |
| Set_Search_Rules -Help |

(optional args) ::= -List_NAME (list name) |
| -No_System |
| -List_NAME (list name) -No_System |

(list name) ::= ATTACH$ |
| COMMAND$ |
| ENTRY$ |
| (legal 32-character name) |

Notes

A list name can be any 32-character string that would be valid in a pathname.

Options are position independent; where options take arguments (like -list_name) the argument must be immediately after the option.
Search Rules in Primos

List names with dollar-signs in them are reserved for Prime use. Users should not create list names that contain dollar-signs.

1.4.2.2 Semantics/Meaning

Pathname

The file containing the search list is typically of the form (anything).(list name).SR. The .SR suffix need not be specified when the pathname is given in the SSR command. This follows the Primos convention that recognized suffices are implicit.

List Names

If the -LIST_NAME option is not given, the list name will be determined by the second-to-last component in the filename. Thus, one could set command search rules either by typing 'SSR COMMAND$.SR -LIST_NAME COMMAND$' or 'SSR COMMAND$.SR'

The list names that attach-scan, the command processor, and the dynamic linker use are, respectively, ATTACH$, COMMAND$, and ENTRY$. Setting a command search list with a list name of MY_COMMAND.SR without explicitly using the -LIST_NAME COMMAND$ option will not achieve the desired result. Doing this would not change the current COMMAND$ list and therefore would have no influence on finding commands.

Any list name is valid even though there may be no program or command that uses the list. This is done because the search rules mechanism has no knowledge of how lists are used. It simply knows what lists have been set. The user can verify that the list was correctly set by using the List_Search_Rules command.

Default Rules

System default rules are kept at SYSTEM>(ENTRY$ COMMAND$ ATTACH$).SR. To obtain the system default rules, the user need not take explicit action; the system default rules will be set up for her. For ENTRY$, the list is established when the first dynamic link is "snapped". For COMMAND$ and ATTACH$, the lists are set up when the user logs in. The ENTRY$ list is not set up upon login because ENTRY$ governs in what order libraries are mapped into the user's address space. If set up at login, it would be too late to change the desired order of mapping.
If a user sets his own rules and later wishes to revert to system default rules, the -DEFAULT option to SSR can be used. When used, all rules in the given list are overwritten with the default rules. Note that the -LIST_NAME option is not necessary although its argument (list name) always is. This is done for compatibility with Rev. 19.4.

If the user sets rules without the -NO_SYSTEM option, the system default list will be inserted in memory before the given list. This has the effect of adding the user's rules to the end of the default list. If the user wishes to completely override the default rules, the -NO_SYSTEM option must be given.

1.4.2.3 Operational Procedures

There are no special procedures for the SSR command.

1.4.2.4 Errors

Standard messages will be printed for the following errors: nonexistent file, unrecognized option, list name longer than 32 characters, no system default rules for this list (when the -no_system is not given).

The Set_Search_Rules command will not directly detect circular references but will attempt to recognize this by limiting the number of iterations to 100.

Keywords are not checked to ensure relevance. For instance, when setting command rules, the occurrence of -added_disks in the list will not be flagged as an error. This is because the search list mechanism should not and does not have knowledge of rules relative to the mechanisms that use the rules. It is the responsibility of the mechanism using the rules (in this case the command processor) to decide how to handle the -added_disks rule.
Search Rules in Primos

1.4.2.5 Restrictions

There are no restrictions other than stated in the Errors section.
1.4.3 List_Search_Rules (LSR) Command

This command causes the search rules mechanism to list the rules in effect for the user. Note that these are the rules in the user's address space, not the rules present in the template file.

The template file, the list name, and the rules for each list are printed. The standard --More-- prompt will be given after each full screen.

1.4.3.1 Syntax/Form

(LSR command) ::= List_Search_Rules {(list name)}8
| List_Search_Rules -Help

Note

{(list name)}8 means zero to eight occurrences of a list name.

1.4.3.2 Semantics/Meaning

If no list names are given, all list rules are listed.

The -HELP option prints only the syntax of the LSR command.

1.4.3.3 Operational Procedures

There are no special procedures for the LSR command.

1.4.3.4 Errors

Standard messages are printed for the following errors: unknown list name, unrecognized option, more than eight lists given, list name of greater than 32 characters.
Search Rules in Primos

1.4.3.5 Restrictions

There are no restrictions other than noted in the Errors section.
1.4.4 Expand (EXP) Command/Function

The Expand command/function is used to expand the name of a file system object using a specified search list. The search list is used to search for the object. If the object is found as a result of the search, the fully qualified pathname of it is reported.

When invoked as a function, Expand will return the fully qualified pathname of the file if found or the string $ERROR$ if the file could not be found.

1.4.4.1 Syntax/Form

(expand command) ::= EXPand (object pathname) {-List_NAME (list name)}
(options)

(options) ::= -SuFFix (legal file system suffix)
{(legal file system suffix)}
| -REFerencing_DIR (pathname)
| -FILE
| -DIRectory
| -SEGment_DIRectory
| -Access_CATegory

1.4.4.2 Semantics/Meaning

**Arguments**

**object pathname**

if object pathname is not fully qualified, it is used to perform the search. If it is qualified, no search is performed and it is returned. (Required)

**list name**

if given, the name of the search list to be used to look for the object. If the search list does not exist, or if the search list argument is not given, then object pathname as given will be used to try to find the file (based on suffices). (Not Required)

**-SuFFix**

if given, optional list of up to 8 suffices to add to the object pathname. (See below for the effects of the suffix list on the search method.) If -suffix is not
given, the system will supply its default suffix list, if possible. The search list name defines this default list, if any. See below for a list of default suffix lists. (Option Not Required.)

-REFerencing_DIR (pathname)
The pathname is substituted for occurrences of [referencing_dir] in the search list. If this option is not given but [referencing_dir] appears as a rule (or part of a rule) in the search list, that rule is skipped. (Option Not Required)

-FILE
If given, search for files. If given with one or more of -directory, -segment_directory, and -access_category, then search only for the types given. (Option Not Required)

-DIRectory
If given, search for directories. Works similarly to -file. (Option Not Required.)

-SEGment_DIREctory
If given, search for segment directories. Works similarly to -file. (Option Not Required)

-Access_CATegory
If given, search for access categories. Works similarly to -file (Option Not Required)

The search method used is the following:

1. For each directory in the search list (or the one given by the object pathname), do the remaining steps.

2. For each suffix in the list of suffices: append the suffix to the entryname of the object pathname unless the object pathname already ends in that suffix. Search the directory for the resulting pathname. If found, then done.

3. Search the directory for the file given by the object pathname with no suffix appended. If found, then return qualified pathname.

The currently available default suffix lists are as follows:

<table>
<thead>
<tr>
<th>List Name</th>
<th>Default Suffix List</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH$</td>
<td>none</td>
</tr>
<tr>
<td>COMMAND$</td>
<td>.RUN, .CPL, .SAVE</td>
</tr>
<tr>
<td>ENTRY$</td>
<td>.RUN</td>
</tr>
</tbody>
</table>

12 October 1984 Functional Specification
1.4.4.3 Operational Procedures

Given above.

1.4.4.4 Errors

If the object is not found, a warning is given. Syntax errors (unrecognized options, missing object pathname, etc.) will be reported.

1.4.4.5 Restrictions

None.
1.4.5 Attach-Scan

Attach-scan is the method by which any file system object that is not fully qualified (i.e., no partition name was given) is found. This method is used, for instance, when attaching to a directory or opening a file. This method would even be implicitly used in conjunction with command search rules if those rules do not specify fully qualified pathnames.

The attach-scan rules are names of partitions only or expressions that evaluate to names of partitions. No pathnames should appear in the attach-scan list.

Attach-scan is an operation of the file system, not the search rules mechanism. The search rules mechanism simply furnishes rules to the file system.

The attach-scan list is called attach$. If a user wanted to follow standard Primos conventions for suffices, the template file would then be called @@.attach$.sr, where @@ means any components can appear before `.attach$.sr`. The system default file has a similar name and the administrator rules are in SYSTEM>ADMIN$.ATTACH$.SR.

1.4.5.1 Syntax/Form

There is no change to the syntax of any commands or system routines because of search-rules in attach-scan.

1.4.5.2 Semantics/Meaning

In the past, an attach-scan was undertaken when an unqualified pathname was given. This attach-scan was a linear search from the beginning to end of the added-disks list. Now, attach-scan will use search rules to qualify the pathname.
1.4.5.3 Operational Procedures

There are no visible operational procedures to attach-scan.

1.4.5.4 Errors

Attach-scan will report errors related to unrecognized keywords that are given as rules.

If the line is down to a given machine on which a remote partition in the search list is located, no error message is returned. This is the same as today. No error is given because this could potentially lead to many errors on every use of attach-scan search rules. Multiple errors would confuse rather than inform the user. The user can find out what partition was attached to by using the LD command etc. or, in a program, by using the GPATH$ procedure. The user can find out the state of the line by using the STAT NETWORK command.

1.4.5.5 Restrictions

There are no restrictions.
1.4.6 Command Search

Command search is the method by which commands are found and executed. This method is used at command level, in CPL programs, and in other types of programs that call CP$.

The command search list is called command$. If a user wanted to follow standard Primos conventions for suffices, the template file would then be called @@.command$.sr, where @@ means any components can appear before '.command$.sr'. The system default file has a similar name and the administrator rules are in SYSTEM>ADMIN$.COMMAND$.SR.

1.4.6.1 Syntax/Form

There is no change to the syntax of any commands or gates because of search-rules in command search.

1.4.6.2 Semantics/ Meaning

Whenever just the name of a command is given, the command search rules will be used to determine the location of the command. If a fully qualified pathname is given as a command, search rules will not be used. If the command is specified in a pathname that begins with the top-level directory (but does not have the partition name), command search rules will not be used but attach-scan rules will be used to fully qualify the name. Attach-scan rules will also be used if the command search rules do not include partition names and thus would not produce a fully qualified name.

1.4.6.3 Operational Procedures

There are no visible operational procedures to command search.
1.4.6.4 Errors

Command search will report errors related to unrecognized keywords that are given as rules.

1.4.6.5 Restrictions

There are no restrictions other than those given in the Errors section.
1.4.7 New Routines

The following routines are Ring 3 gates. The first two are for opening a file with search rules. The rest make up the interface to the search rules mechanism.

Not all routines are used by Primos, but users may find them useful. Each routine is described in its own section.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPN$SR</td>
<td>Open an object using search rules.</td>
</tr>
<tr>
<td>OPN$SRSF</td>
<td>Open an object using search rules and suffixes.</td>
</tr>
<tr>
<td>SR$ADDB</td>
<td>Add rule to a list before an existing rule.</td>
</tr>
<tr>
<td>SR$ADDE</td>
<td>Add rule to the end of a list or after an existing rule.</td>
</tr>
<tr>
<td>SR$CREAT</td>
<td>Create list by name.</td>
</tr>
<tr>
<td>SR$DELET</td>
<td>Delete a search list specified by name.</td>
</tr>
<tr>
<td>SR$FR_LS</td>
<td>Free storage used to hold a search list.</td>
</tr>
<tr>
<td>SR$LIST</td>
<td>Return a list of all search list names in this process.</td>
</tr>
<tr>
<td>SR$NEXTR</td>
<td>Return the next rule from a given list.</td>
</tr>
<tr>
<td>SR$READ</td>
<td>Return a list of all rules of a given list.</td>
</tr>
<tr>
<td>SR$REM</td>
<td>Remove a search rule from a list.</td>
</tr>
<tr>
<td>SR$SETL</td>
<td>Set the locator value in a given rule.</td>
</tr>
<tr>
<td>SR$TEMPL</td>
<td>Process a list template file.</td>
</tr>
</tbody>
</table>
Opening a File with Search Lists

1.4.8 OPN$SR - Open an Object Using Search Rules

Syntax/Form

Usage:  
call opn$sr(list_name, referencing_dir,  
       file_path, open_mode,  
       types, found_path,  
       out_unit, out_type,  
       code);

Declaration:  
dcl opn$sr entry(char(32) var, char(128) var,  
                  char(128) var, fixed bin,  
                  fixed bin, char(128) var,  
                  fixed bin, fixed bin,  
                  fixed bin);

1.4.8.2 Semantics/ Meaning

Given the name of a search list and the name of a file, 
OPN$SR opens the file. If the filename is not an entryname, 
i.e., it contains '>'s, then that specific file is opened if possible; 
the specified search rules are not used to find it. 
If the filename is an entryname, a pathname is formed with 
each rule in the list, in turn. If that pathname is still not 
fully qualified, then rules for attach-scan are implicitly 
employed to obtain a fully qualified name.

OPN$SR cannot be used to create files. That is, if the 
major key supplied in open_mode is k$writ or k$rdwr, and the 
file does not yet exist, it will not be created. This is 
because it would not be possible to determine in which of the 
search directories to create the file.

Example 1

/* No search rules used in this call. */
call opn$sr(test, ..., <osgrp1>alan>my_file, ...);

Example 2

/* Search rules from list with list name TEST are used. */
call opn$sr(test, ..., my_file, ...);
Search Rules in Primos

If TEST is composed of these rules:

\(<\text{osgrp1}>\text{alan}\>
\(<\text{osgrp2}>\text{alan}\>

then only the rules in TEST are used and the file
\(<\text{osgrp1}>\text{alan}>\text{my\_file}\) is opened.

If TEST is composed of these rules:

\(<\text{osgrp1}>\text{alan}\>
\(<\text{osgrp2}>\text{system}\>

and ATTACH$ is composed of these rules:

\(<\text{osgrp1}>\>
\(<\text{osgrp2}>\>

then ATTACH$ rules must be employed to fully qualify the pathname. First, the pathname \(<\text{osgrp1}>\text{alan}>\text{my\_file}\) would be constructed. Then the first ATTACH$ rule would be used to construct \(<\text{osgrp1}>\text{alan}>\text{my\_file}\). If that file could not be opened, \(<\text{osgrp2}>\text{alan}>\text{my\_file}\) would be tried. If that failed, \(<\text{osgrp1}>\text{system}>\text{my\_file}\) would be used, then \(<\text{osgrp2}>\text{system}>\text{my\_file}\). If the last of these fails an error code of "Not found" is returned. That code is only returned if all of the rules were unsuccessful.

Example 3

/* Only ATTACH$ rules used to find file in the call. */
call opn$sr(test, ..., \text{alan}>\text{test\_file}, ...);

Rules from the list TEST were not used because the given pathname was not just an entryname; there were ">"s in the name. The pathname \(<\text{osgrp1}>\text{alan}>\text{my\_file}\) is opened.

Arguments

\text{list\_name} 
the name of the search list to be used. If \text{list\_name} is the null string or if that search list does not exist, then no search list is used; \text{file\_path} is used "as-is" to find the file. (Input)

\text{referencing\_dir} 
the pathname of the directory containing the object making the search request. For example, in the case of a statement in a PL1 program, the directory containing the source file would be the \text{referencing\_dir}. This value is used only to process search rules of the form "[\text{referencing\_dir}]\text{path}". If \text{referencing\_dir} is null, then any such search rules are simply skipped. (Input)
file_path

usually the entryname of the file to search for. If file_path is not an entryname, i.e., if it contains any ">"s, no search is made, but an attempt is made to open that specific file. This avoids applications programs having to special case for this. (Input)

open_mode

a subset of the request keys of SRCH$$. Legal major (action) keys are: k$exist, k SceneManager, k$read, k$writ, k$rdwr, k$vmr; Legal minor (ref) keys are k$getu. No newfil keys are allowed. (Input)

types

types of objects that should be found as a result of the search. These types are denoted by new additive keys:

<table>
<thead>
<tr>
<th>key</th>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>k$cat</td>
<td>access category</td>
<td>'00010'b</td>
</tr>
<tr>
<td>k$file</td>
<td>file</td>
<td>'00100'b</td>
</tr>
<tr>
<td>k$sd$</td>
<td>segdir</td>
<td>'01000'b</td>
</tr>
<tr>
<td>k$dir</td>
<td>directory</td>
<td>'10000'b</td>
</tr>
</tbody>
</table>

found_path

fully qualified pathname of the file actually opened. (Output)

out_unit

unit number upon which to open the file, or upon which the file was opened if k$getu was used. (Input or Output)

out_type

type of the object successfully opened (same values as from SRCH$$. (Output)

code

standard file system error code. (Output)

Operational Procedures: None.

Errors: Given above.
Search Rules in Primos

Restrictions: OPN$SR cannot be used to create a file.
1.4.9 OPN$SRSF - Open an Object Using Search Rules and Suffices.

Syntax/Form

Usage: call opn$srsf(list_name, file_path, suffix_list_ptr, n_suffices, open_mode, referencing_dir, suffix_index, basename, found_path, unit, type, code);

Declaration: dcl opn$srsf entry(char(32) var, char(128) var, ptr, fixed bin, fixed bin, char(128) var, fixed bin, char(128) var, fixed bin, fixed bin);

Semantics/ Meaning

OPN$SRSF is used to search for a file using both a search list and Primos suffix conventions. This primitive is similar to OPN$SR in its use of search lists and open_mode. The difference lies in the application of the suffix conventions as explained below.

Like OPN$SR, OPN$SRSF cannot be used to create files.

Arguments

list_name
the name of the search list to be used. If list_name is the null string or if that search list does not exist, then no search list is used; file_path is used "as-is" to find the file. (Input)

file_path
if an entry name, file_path specifies the file to be searched for. Otherwise, the search list is not used, and only the specific directory given by file_path is searched. The search method is as follows:

1. for each directory in the search list (or the one given by file_path), do the remaining steps.
2. for each suffix in the suffix_list: append the suffix to the entryname of file_path unless file_path already ends in that suffix. Search the directory for the resulting file name. If found, then finished.

3. search the directory for the file given by file_path with no suffix appended. If found, then finished.

If the file is not found by the steps above, then it is not in the search path. (Input)

suffix_list_ptr
pointer to an array of filename suffices. The array must have the datatype "char(32) var", and its lower bound must be 1. The suffices will be used as explained under "file_path" above. If no suffices are desired, either use OPN$SR or pass the value null() for suffix_list_ptr; in the latter case, n_suffices must be 0. (Input)

n_suffices
the number of elements (suffices) in the array pointed to by suffix_list_ptr. If n_suffices is 0, then OPN$SR SF acts just like OPN$SR (no suffix processing done). (Input)

open_mode
same as for OPN$SR. (Input)

referencing_dir
same as for OPN$SR. (Input)

suffix_index
valid only if n_suffices is nonzero. If valid, it is the index (counting from 1) in the suffix_list of the suffix using which the file was found. A value of 0 means that file_path did not end in one of the suffices, and that the name of the file found also did not end in one of the suffices. The suffix_index is validly returned even if the file_path already ended in the specified suffix. (Output)

basename
the entryname of the file found, minus the suffix given by suffix_index. (Output)

found_path
the fully qualified pathname of the file actually found. This pathname contains the suffix specified by suffix_index. (Output)
Search Rules in Primos

unit
same as for OPN$SR. (Input or Output)

type
same as for OPN$SR. (Output)

code
standard file system error code. (Output)

Example

dcl suffix_list(2) char(32) var static init
(‘.INS.PLP’, ‘.INCLUDE.PLP’);

... call opn$srsf(‘INCLUDE’, ‘FOO’, addr(suffix_list), 2,
  k$read, ref_dir, suffix_index, found_path,
  basename, unit, type, code);

In this example, OPN$SR$SF might find the file FOO.INCLUDE.PLP
in one of the directories of search list INCLUDE. In this case, suffix_index would be 2, and the basename would be
"FOO".

Operational Procedures: None.

Errors: Given above.

Restrictions: OPN$SR$SF cannot be used to create files.
1.4.10 SR$CREAT - Create a Search List

Syntax/Form

Usage: call sr$creat(list_name, code);

Declaration: dcl sr$creat entry(char(32) var, fixed bin);

Semantics/ Meaning

SR$CREAT creates a new search list with the given list name if that list does not already exist. If the list already exists, the call performs no operation; the list is left intact. The code indicates whether the list existed before the call.

If there are administrator rules for the given list, they will be included in the list upon creation.

Arguments

list_name

the name of the search list to be created or found if it already exists. (Input)

code

standard error code. Its value is 0 if no errors occurred, E$EXIST if the list already existed, and E$ROOM if the list could not be created because of lack of storage. If E$ROOM is returned, the state of the user's search rules is the same as before the call. (Output)
Search Rules in Primos

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
1.4.11 SR$DEL - Delete a search list.

Syntax/Form

Usage: call sr$del(list_name, code);
Declaration: dcl sr$del entry(char(32) var, fixed bin);

Semantics/ Meaning

SR$DEL deletes the given search list if it exists. If it does not exist, an error code is returned.

Arguments

list_name
- name of the list to be deleted. (Input)

code
- standard error code. Its value is 0 if the list was successfully deleted, and E$FNTF if no list was found with the given list name.

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
Search Rules in Primos

1.4.12 SR$INIT - Initialize a Users Search Lists.

Syntax/Form

Usage: call sr$init(code);

Declaration: dcl sr$init entry(fixed bin);

Semantics/meaning

SR$INIT is used whenever the user's Ring 3 environment is initialized. It deletes any existing search lists, then creates default lists COMMAND$ and ATTACH$ (includes administrator and system rules) for the user. The default ENTRY$ list is not created until the first link is "snapped" and libraries are mapped in. This gives the user a chance to map in libraries in an order different from that given by the default rules. To do this, the user would issue a SSR (pathname of ENTRY$ list) early in login.cpl.

All search lists are automatically created with any administrator rules that might be present.

Arguments

code

standard error code. Its value is 0 if no errors occurred and E$ROOM if there was not sufficient space to create the necessary lists. If E$ROOM is returned, some lists may have been created but there is no guarantee about how many or which ones.

Operational Procedures: None.

Errors: Given above.
Search Rules in Primos

Restrictions: None.
Search Rules in Primos

Adding and Removing Search Rules in Lists

1.4.13 SR$ADDDB - Add Rule Before a Given Rule

Syntax/Form

Usage:
call sr$addb(list_name, old_rule,
              new_rule, code);

Declaration: dcl sr$addb entry(char(32) var, char(128) var,
                              char(128) var, fixed bin);

Semantics/Meaning

SR$ADDDB is used to add a new search rule before an existing rule in an existing search list.

SR$TEMPL should be used to process an entire template file.

Note that function rules and their current values are not considered the same rule. For example, even if the current home directory is A>B>C, the two rules "[home_dir]>xyz" and "a>b>c>xyz" are different rules.

This primitive will not allow rules to be added before any administrator rules.

Arguments

list_name
  name of the search list to which the rule is to be added. (Input)

old_rule
  rule before which new_rule is to be added. If old_rule is the null string ('"'), new_rule is added to the beginning of the search list. (Input)

new_rule
  rule to be added. This rule must not be a -system, -eval, or -use rule; these rules cannot be added via SR$ADDDB or SR$ADDE, as they can appear only in template files. (Template files are processed with SR$TEMPL.)
Search Rules in Primos

(Input)

code
standard error code. Its value is 0 if the new rule was successfully added, E$FNTF if the old rule was not found, E$BPAR if either the old rule or the new rule is not in the correct format, E$ROOM if there was insufficient storage to add the new rule, and E$ADMN if a rule was to be added before an administrator rule (new error code). (Output)

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
1.4.14 SR$ADDE - Add Rule to End (After Given Rule)

Syntax/Form

Usage: call sr$adde(list_name, old_rule, new_rule, code);

Declaration: dcl sr$adde entry(char(32) var, char(128) var, char(128) var, fixed bin);

Semantics/ Meaning

SR$ADDE works like SR$ADDB except that the new rule is added after the old_rule instead of before.

Note that function rules and their current values are not considered the same rule. For example, even if the current home directory is A>B>C, the two rules "[home_dir]>xyz" and "a>b>c>xyz" are different rules.

This primitive will not allow a rule to be added before an administrator rule.

Arguments

list_name
name of the search list to which the rule is to be added. (Input)

old_rule
rule after which new_rule is to be added. If old_rule is the null string, new_rule is added to the end of the search list. (Input)

new_rule
rule to be added. (Input)

code
standard error code. Its value is 0 if the new rule was successfully added, E$FNTF if the old rule was not found, E$BPAR if either the old or new rule is not in the correct format, E$ROOM if there was insufficient storage to add the new rule, and E$ADMN if a rule was to be added before an administrator rule. (Output)
Search Rules in Primos

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
Search Rules in Primos

1.4.15 SR$REM - Remove a Rule

Syntax/Form

Usage: call sr$rem(list_name, rule, code);

Declaration: dcl sr$rem entry(char(32) var, char(128) var, fixed bin);

Semantics/ Meaning

SR$REM is used to remove a search rule from an existing search list.

Note that function rules and their current values are not considered the same rule. For example, even if the current home directory is A>B>C, the two rules "[home_dir]>xyz" and "a>b>c>xyz" are different rules.

This primitive will not allow the removal of an administrator rule.

Arguments

list_name
name of the search list from which a rule is to be removed. (Input)

rule
text string of the rule to be removed. (Input)

code
standard error code. Its value is 0 if the rule was successfully removed, E$FNTF if the rule was not found in the given search list, and E$ADMN if the rule to be removed was an administrator rule. (Output)
Search Rules in Primos

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
1.4.16 SR$TEMPL - Process a Template File

Syntax/Form

Usage: call sr$templ(template_path, list_name, overwrite, error_path, error_line, code);

Declaration: dcl sr$temp entry(char(128) var, char(32) var, bit(1) aligned, char(128) var, fixed bin, fixed bin);

Semantics/meaning

This primitive is given the pathname of a search list template file, and the name of a search list. SR$TEMPL processes all rules in the template file and adds them to the search list. If an error occurs during processing, no rules are added to the search list; it remains in the same state as before the call. If there are rules in the list before the call, the overwrite flag determines whether the new rules will overwrite the existing ones or be added after them. If the new rules are to overwrite, the locator values will be copied from old rules to matching new rules. If overwrite is TRUE, administrator rules will be included in the list before the rules in the specified pathname.

The keywords -use and -system are processed by processing the template files to which they point. Nesting up to 100 levels is allowed.

The caller should use error_path and error_line to print a message for the user.

Arguments

template_path
pathname of the template file to be processed. Read access to the file is required. (Input)

list_name
name of the search list to which the rules in the template are to be added. List name will be used to locate and include the system default and administrator template files, if needed. (Input)
override flag that determines whether the new rules from the template file will overwrite any existing rules or be added after them. If override is TRUE, locator values from old rules will be copied to matching new rules.

error_path pathname of the template file in which an erroneous rule was found. Error_path can differ from template_path when that template file contains a -use or -system rule, and it is one of those files that is in error. (Output)

error_line line number within error_path of the erroneous rule. (Output)

code standard error code. Its value is 0 if the template file was processed successfully, E$ROOM if there was insufficient storage to hold the search list, E$BPAR if a rule (identified by error_path and error_line) was in incorrect format, E$FNTF if a template file referenced by a -use or -system rule could not be found. (Other file system errors may be returned as well, e.g., E$NRIT for insufficient access rights.) If any error occurred, the list indicated by list_name remains the same as before the call.

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
Performing a Search

1.4.17 SR$NEXTR - Get Next Rule

Syntax/Form

Usage: current_rule_ptr = sr$nextr(list_name, referencing_dir, prev_rule_ptr, locator, search_place);

Declaration: dcl sr$nextr entry(char(32) var, char(128) var, ptr, char(128) var) returns (ptr);

Semantics/ Meaning

SR$NEXTR is used to fetch the pathname of the next place to search as given by a specified search list. It is intended to be used by procedures using a search list to conduct a search for an object, so that they are independent of the format of search lists in memory. To read the first rule in the list, prev_rule_ptr must have the value null(). Subsequent calls give the actual prev_rule_ptr and a pointer to the next rule is then returned.

Arguments

list_name
- name of the search list being used to search for the object. (Input)

prev_rule_ptr
- pointer to the previous rule (the one the caller just used and with which it failed to find the object) in this search list. The value null() should be supplied in order to start with the first rule in the list. (Input)
Search Rules in Primos

referencing_dir
pathname of the directory containing the object making
the reference. This pathname will be used to process
"[referencing_dir]" search rules. For example, in the
case of the processor for the PL1 " directory
containing the source file which has the " statement in
it. If referencing_dir is the null string, all
"[referencing_dir]" rules will be ignored by sr$nextr.
(Input)

locator
Locator value saved for this rule by a previous call to
SR$SETL. The locator can be used by a search procedure
to remember a correspondence between the pathname of
this search rule and the exact place to search. For
example, it could be used to save a pointer to the EPF
corresponding to an ENTRY$ search rule. If the locator
for this rule has never been set by a call to SR$SETL,
null() will be returned in locator. (Output)

search_place
pathname of the place to search corresponding to this
rule. If the rule in question is "[home_dir]", then
the string "*" is returned for search_place. This
enables the search procedure to _concatenate
">entryname" and obtain "*>entryname", which is the
correct pathname. For all other rules, a complete (but
not necessarily absolute) pathname is returned.
(Output)

current_rule_ptr
pointer to the rule the data for which has just been
obtained. The value of current_rule_ptr should be
returned to SR$NEXTR as previous_rule_ptr in the next
pass through the search loop. Current_rule_ptr will be
null() when there are no more rules in this search
list.

Operational Procedures: None.

Errors: Given above.
Search Rules in Primos

Restrictions: None.
Search Rules in Primos

1.4.18 SR$SETL - Set Locator

Syntax/Form

Usage: call sr$setl(rule_ptr, locator, code);
Declaration: dcl sr$setl entry(ptr, ptr, fixed bin);

Semantics/Meaning

SR$SETL is used to set the locator value for a specified search rule. The locator value is a short pointer; it can be set to any value the caller desires. A typical use is to set the locator to point to an EPF corresponding to an ENTRY$ search rule.

Note that it is the responsibility of the caller to ensure either that the locator value set by sr$setl remains valid (e.g., that the EPF is not removed from the address space), or that the procedure that invalidates the locator remembers to call SR$SETL to update the locator in the search rule.

This primitive cannot be used to set a locator for an administrator rule.

Arguments

rule_ptr
pointer to the search rule whose locator value is to be set. This pointer must be obtained from a call to SR$NEXTR. (Input)

locator
locator value desired for this rule. A value of null() may be set to indicate an invalid locator. (Input)

code
standard error code. Its value is 0 for a successful operation or E$ADMN if the rule was an administrator rule. (This is a new argument since rev. 19.4)
Search Rules in Primos

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
Search Rules in Primos

Showing Search Lists

1.4.19 SR$LIST - Get Names of Defined Lists

Syntax/Form

Usage: call sr$list(output_ptr, code);
Declaration: dcl sr$list entry(ptr, fixed bin);

Semantics/ Meaning

SR$LIST returns a list of the names of all search lists currently defined in the calling process. It is primarily useful in applications like the list_search_rules command.

Note that when the user is finished with the returned list, he is expected to call SR$FR_LS (see below) to free it up.

Arguments

output_ptr
  pointer to a linked list structure, which the caller can walk with a simple do-repeat loop. The format of an entry in the list is:

  dcl 1 list_ent based,
  2  version fixed bin,
  2  length fixed bin,
  2  next ptr options(short),
  2  list_name char(32) var;

version
  version number of the structure. In this release, its value is 0.

length
  length in bytes of the entire structure. In this release, its value is 41.

next
  pointer to the next item in the list of names.
list_name
   name of the search list represented by this entry on the list.

code
   Currently, code always has the value 0.

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
1.4.20 SR$READ - Read the Rules in a List

Syntax/Form

Usage: call sr$read(list_name, output_ptr, code);

Declaration: dcl sr$read entry(char(32) var, ptr, fixed bin);

Semantics/ Meaning

SR$READ is used to return the character string representations of all rules in a given search list. This primitive is most useful in an application like the list_search_rules command.

Note that when the user is finished with the returned list, he is expected to call SR$FR_LS (see below) to free it up.

Arguments

list_name
name of the list to be read. (Input)

output_ptr
pointer to a linked list structure containing the character representations of all the rules in the list. The caller is expected to walk this list structure with a simple do-repeat loop in order to access all the members. The format of each member of the list is:

dcl 1 output_ent based,
  2 version fixed bin,
  2 length fixed bin,
  2 next_ptr options(short),
  2 rule_str char(128) var;

version
version number of this structure; for this release, the value is 0.

length
length of the structure in bytes; for this release, the value is 137.
Search Rules in Primos

next
  pointer to the next rule in the list.
rule_str
  character string representation of the rule.

code
  standard error code. It will be E$ROOM if sufficient
  system free storage could not be allocated to hold the
  output structure.

Operational Procedures: None.

Errors: Given above.

Restrictions: None.
Search Rules in Primos

1.4.21 SR$FR_LS - Free List Structures

Syntax/Form

Usage: call sr$fr_ls(structure_ptr);
Declaration: dcl sr$fr_ls entry(ptr);

Semantics/Meaning

This routine is called by the user to free the linked list structures returned by both SR$LIST and SR$READ.

Arguments

structure_ptr
structure to be freed by SR$FR_LS. SR$FR_LS does not know the format of the structure except that the third word is the pointer to the next structure; the last structure in the list has a null() pointer.

Operational Procedures: None.

Errors: Given above.

Restrictions: None.

2 Testability

Several factors increase the testability of the Search Rules project: most of the project involves Ring 3 code, interfaces are well defined, and areas of Primos to be changed are reasonably well written.

In addition, the actual search rules routines have been in use in the Bootleg Command Environment since 1981.
Search Rules in Primos

3 Design Notes

As mentioned above, circular references in search lists will be detected only by limiting the number of iterations (of references) to 100. Direct detection of circular references is not done because of the complexity involved.

4 Reliability

This project should neither increase nor decrease the reliability of the system. No increase in reliability is sought.

5 Performance

This project allows users to better configure their environment to maximize performance in attach-scan. They need only order partitions in their attach-scan search rules in the order of most to least frequently referenced.

A user could improperly configure his search rules so that performance is minimized. Proper documentation will alleviate this problem.

6 Configuration

Search rules require no special hardware configuration.

7 Installation

Two files must exist on the master disk when it is shipped: SYSTEM>ATTACH$.SR and COMMAND$.SR. ATTACH$.SR contains the single rule -ADDED_DISKS. COMMAND$.SR contains the single rule CMDNCO. These two files ensure complete compatibility with existing systems and configurations. There will be no administrator rules on the master disk.
8 Ease of Use

Search rules are intended for the experienced user who wishes to tailor her environment for better performance (quicker attach-scans) and more flexibility (execute commands from any directory). In addition, the designer of programs and sub-systems may also choose to use search rules for added flexibility.

9 Maintainability

This project will neither increase nor decrease the maintainability of Primos. Since the areas of Primos to be modified in this project are reasonably well written, no unusual effort will be made to increase their maintainability.

10 Compatibility

The files shipped on the master disk assure that Primos appears compatible for all users who do not set their own search rules. System administrators who are concerned about compatibility for accounting, etc. software may want to create administrator rules (see above) to force compatibility.

11 Standards

The OS Coding Guidelines and NPIP will be followed.

12 References

A functional specification written in 1981 by Brad Hampson details the search rules routines. This document is available from Alan Dossett.