To: Distribution

From: Paul Smee

Date: January 11, 1978

Subject: Command Search Facility

INTRODUCTION

This memorandum describes the initial implementation of the command search facility. This facility was originally proposed by Tom McGary in MTB-112, and the proposal was revived by Bill Silver in MTB-336X. The initial implementation is a modified subset of the facility proposed by Bill Silver. The differences are summarized in the section SIGNIFICANT MODIFICATIONS FROM MTB-336X, below.

This new command search facility will greatly enhance the already powerful Multics capability to dynamically customize a user's environment. It consists of a set of commands and subroutines that will initialize, maintain, and search lists of pathnames. Any Multics command that needs its own per-process list of pathnames can use this new facility.

This memorandum contains sections discussing the following subjects:

- need for a command search facility
- overview of the proposed new search facility
- MPM command documentation
- MPM subroutine documentation

Please send all comments and suggestions on this memorandum to the author.

Send mail to: Paul Smee
Honeywell Information Systems
575 Tech. Sq.
Cambridge, Mass. 02139

or send Multics mail at M.I.T or System M to:

Smee.Multics

or call: (617) 492_9308
HVN 261-9308

SIGNIFICANT MODIFICATIONS FROM MTB336X

Originally the search facility was going to be implemented as described in MTB-336X, but certain logical inconsistencies were detected during design and coding. Therefore it was decided to implement a modified subset of the originally proposed facility, in order that a search facility might be available for use by the WORDPRO software. The current implementation has been designed to allow future extension into a facility approximating the originally proposed facility.

The most significant change is that the automatic initialization of search lists has not been implemented. The user must explicitly initialize any search lists which are to be used. This restriction can be removed after certain difficulties are ironed out.

As a result of the change given above, the idea of a generalized search segment is meaningless. Only the Process Search Segment is defined and used.

Also as a result of the change given above, the -segment control argument is meaningless and has been removed from all commands.

All subroutines which return a filled in structure to the user have been modified so that the caller must supply an area into which the search facility can allocate the structure, rather than a copy of the structure itself. This was done because there is no easy way for the caller to know how long the structure must be, while the necessary information to determine size is available to the search facility.

A new entry point (search_paths_$find_all) has been added. This entry finds all occurrences of a specified pathname using the given search list. This complements the original find_dir entry, which returned only the first occurrence found.

The old entry points get and get_unexpanded have both been replaced by a single new entry point, search_paths_$get. Get offers an increased amount of flexibility, as it uses a set of control bits to determine which types of pathnames should be expanded. Thus, it can perform all the functions of the original two entries, and more.

Two new entry points have been added which allow a calling subroutine to delete specific paths from a given search list, and to delete a given search list from the process search segment. These are delete_list and delete_path.

NEED FOR A COMMAND SEARCH FACILITY

The Concept of Searching

The concept of searching for an object in an ordered, per-process list of places is one that is fundamental to Multics. The Multics linker provides an example of its complete implementation. The objects that the linker searches for are "object segments". The places that it searches are "directories". In addition to the search capabilities itself, a complete set of commands is provided to manipulate the per-process list of directories to be searched. (1)

Dynamic linking provides Multics users with a flexibility and an "interactive feel" simply not available with any other operating system. Furthermore, the capability to dynamically control his own list of directories being searched, gives the Multics user an unparalleled ability to dynamically customize his executable address space.

Current Command Searching Problems

Several Multics commands currently use, to varying and limited degrees, the concept of searching. (2) Each of these commands implements its own special case search facility. Currently there is no generalized search facility that can be conveniently used by all commands. The absence of a generalized search facility has resulted in many problems. A list of these problems and examples of commands that suffer from these problems is given below:

1. Searching Not Performed: A number of Multics commands that should use a dynamic search facility, currently do not. The reason they do not is that no general search facility is available for them to use, and no command implementer wants to go through the trouble to implement a complete search facility for just one command. An example of a Multics command that should, but does not, use a search facility is "exec_com". Because "exec_com" does not search for the ".ec" file that it executes, it is virtually impossible for a project, subsystem, or user to establish an "exec_com" library that can be used conveniently.

---

(1) A complete description of the linker's search facility is given in MTB-112.

(2) Descriptions of the search facilities provided by several Multics commands are given in MTB-112.

2. Incomplete Implementations: None of the search facilities currently used by Multics commands have been implemented as completely as that of the Multics linker. For example, the "translator" search facility, as implemented by the commands that set and print translator search rules, does not provide the dynamic updating capabilities provided by the commands that add and delete linker search rules.

3. Inefficient Searching: Some Multics commands use search facilities that were specifically designed for other commands. For example, the runoff command uses a search facility that was designed for the Multics language translators. This often results in runoff searching for ".runoff" insert files in directories containing large numbers of language processor include files, but no ".runoff" insert files.

4. Not Easy To Learn: Because several Multics commands each use different search facilities, it is more difficult for a Multics user to learn to use any of them. A user that has learned how to use the "translator" search facility will have learned nothing about the search facility provided by the "teco" text editor.

5. Not Easy To Use: The lack of a generalized search facility makes it more difficult to initialize all of the search facilities that will be used by a process. In order to avoid the inefficiencies involved when several commands use the same specialized search facility, users must often set and reset the search facility when switching from one command to another.

Conclusions:

The concept of searching, as applied to the Multics dynamic linking facility, has proven to be a very powerful and useful concept. However, as applied to the Multics commands, the concept of searching has either been applied poorly, or not at all.

What is needed is a generalized and complete search facility that can be used by any Multics command. Multics commands that need a search facility will have one readily available, with no extra implementation effort, and one that Multics users already know how to use. By applying the concept of searching to the many Multics commands that need it, the Multics user will be given an even more powerful capability to dynamically customize his process environment.

OVERVIEW OF THE PROPOSED SEARCH FACILITY

This memorandum proposes the implementation of a generalized search facility that can be used by any Multics command. It will maintain, for any command, the command's own per-process list of places where the command should search to find the objects that it deals with.

Important Features of the New Search Facility

1. The new search facility is not intended to replace the existing linker search facility. It provides and additional search capability for commands.

2. The new search facility will provide users with a complete set of commands that initialize, update, and print their current search information for any and all commands. These search facility commands have deliberately been designed to resemble the existing commands that add, delete, print, and set linker search rules.

3. The new search facility will provide a complete set of subroutine interfaces that allow commands to access their search information. Special consideration was given to providing commands with an efficient mechanism for determining when their search information has been modified.

4. The new search facility places no restrictions on the places to be searched. They may be directories, but they may also be segments or files or archives, etc.

5. Since it is very common for directories to be the places to search, and for segments to be the objects searched for, the new search facility will provide a command and subroutine interface to find a segment in a specified list of directories.

6. The new search facility does not provide for the initiation of segments. This is left up to the commands that use the search facility. They are better able to decide whether initiation is necessary, and if so, whether a reference name should be initiated, a bit count returned, a copy made, etc.

7. The new search facility uses its own capabilities to initialize the search information for a process. The full power and flexibility of the search facility is used to find the data needed to initialize this search information. The same commands that are used to manage the search information are used to manage this initialization data. No special search facility ASCII "source" language is needed.

Definition of Terms

In order to more easily understand the MPM documentation presented later in this memorandum, the following terms will be defined:

1. Search Path: A "search path" identifies a "place" to be searched. The "place" may be a directory, file, or segment. A search path may be represented as either an absolute pathname or one of the following keywords:

"unexpanded pathname"
is an absolute pathname that contains the active function [user name] or [user project]. Such a pathname will be expanded once per process. For example, the search path ">udd>[user project]" could be used to define the project directory of all users. When this search path is expanded in a user's process it will yield the project directory of that user.

-stop
is a keyword that does not represent a pathname. It is used during the initialization of search paths. It is ignored at all other times. The function of this keyword during search path initialization is described in the next section.

-referencing_dir
is computed each time it is used.

-working_dir
is computed each time it is used.

-process_dir
is computed once per process.

-home_dir
is computed once per process.

2. Search List: is a set of search paths. Search lists are identified by name. Any valid Multics entryname is accepted. It has been suggested that search list names correspond to the "suffix" name associated with the command that will use the search list. However, the search facility will not enforce this rule. It is up to the implementor of a command to select and promulgate the name of the search list that will be used by that command.

3. Search Segment: A search segment contains one or more search lists. All search lists residing in the user's search segment must have different names.

4. Process Search Segment: contains the current, temporary representations of all search lists that have been referenced by the process. (1) The use of process search segments will be transparent to the user. These search segments will reside in the process directory. In the present implementation, the terms Process Search Segment and Search Segment are interchangible.

Benefits of the New Command Search Facility

1. Expanded Use of Searching: It is expected that many Multics commands that currently do not use searching will be upgraded to use the new search facility. Once the new search facility is available, this can be done with a minimum of implementation effort. This will result in an improved user interface and in enhanced functional capability for these commands. The next section presents a list of commands that may use the new search facility.

2. Complete Implementation: The new search facility will provide a complete set of command and subroutine interfaces for the management of search lists.

3. Efficient Searching: Each command that performs searching will be able to have its own search list. This will eliminate wasteful searching in places that do not contain the type of object that the command is searching for. In addition, a mechanism is provided that will allow a command or subroutine to efficiently determine when its search list has been changed.

---

(1) The actual implementation may involve keeping the current process search information in system free areas rather than in a dedicated search segment.

4. Efficient Storage: Each process will have to initialize and copy into its process search segment only those search lists that it actually uses.

5. Easy to Learn to Use: The new search facility is easy to learn to use because it works the same way for all search lists and thus for all commands that use search lists.

6. Eliminate the Home Directory Syndrome: There are Multics commands that do not know where to find a segment that is their job to use. These commands currently solve this problem by looking in the user's home directory. The new search facility provides a mechanism for these commands, and the users of these commands, to conveniently override this restrictive default convention.

7. Provide a Push and Pop Facility: There are Multics subsystems that can be told to use a particular segment over a number of invocations. They will use that segment until told to use a different one. However, a new segment can not be used on a temporary basis without remembering what previous segment was in use, and ther telling the subsystem to use the previous segment again when the temporary segment is no longer wanted. This is usually inconvenient and often virtually impossible. The new search facility provides a mechanism that allows a subsystem to push and pop the segment currently in use.

Users Of The New Search Facility

Presented below is a list of current Multics commands and subsystems that would benefit from using the new search facility:

Language Processors: Many of the Multics language processors (PL/I, FORTRAN, COBOL, ALM) use searching. With the new search facility each could have its own search list. They could, however, continue to share the "translator" search list. In this case, the current commands that set and print the "translator" search list would be converted to use the new search facility.

exec_com: The new search facility will make it possible to conveniently use libraries of exec_coms. Private versions of exec_com that implement their own searching would be unnecessary.

Text Editors: The Multics text editors that provide macro capabilities (edit, qedx, teco) could use the new

search facility to make it convenient to use editor macro libraries.

Help: The "help" command could use the new search facility to find "info" segments. The need for this enhancement to the "help" command can easily be demonstrated by pointing out the fact that several Multics projects have implemented their own help command just to perform searching.

Compose: The "compose" command will use its own search list to find insert files.

Dictionary Commands: The new word processing commands for spelling error detection and hyphenation with use the new command search facility to specify lists of dictionaries.

Multics System Tapes: The generate_mst command could use the new search facility to specify the directory to be searched when loading segments onto a Multics system tape.

Home Directory Dependants: The "debug" command could use the new search facility to specify the default location for "break segments".

Abbrev: The "abbrev" subsystem could use the search facility to push and pop the current profile segment being used. This would allow exec_com written subsystem to conveniently replace a user's "profile" on a temporary basis. Currently the absence of this capability often causes exec_com written subsystems to malfunction due to some user defined abbreviation.

MPM Documentation

The remainder of this memorandum presents draft MPM documentation of the commands and subroutines that implement the proposed new command search facility.

Name: add_search_paths, asp

The add_search_paths command adds one or more search paths to the specified search list.

Usage

add_search_paths search_list search_paths

where:

1. search_list
   is the name of the search list to which the new search paths is added.

2. search_path
   specifies a new search path and its position within the search list. A search path is specified as follows:

   new_search_path {-control_arg}

   where new_search_path is a relative or absolute pathname or a keyword. For a list of acceptable keywords see the notes below. The control argument can be chosen from the following:

   -after cur_search_path, -af cur_search_path
   specifies that the new search path is positioned after the current search path. The current search path is an absolute or relative pathname or a keyword. In representing the current search path it is necessary to use the same name that appears when the print_search_paths command is invoked (without the -exp option).

   -before cur_search_path, -be cur_search_path
   specifies that the new search path is positioned before the current search path.

   -first, -ft
   specifies that the new search path is positioned as the first search path in the search list.

-last, -lt
specifies that the new search path is positioned as the last search path in the search list. If no search path position control argument is specified, then -last is assumed.

Notes

Listed below are the keywords accepted as search paths in place of absolute or relative pathnames. There is no restriction as to the position of any of these keywords within the search list.

```
-home_dir
-process_dir
-referencing_dir
-working_dir
-stop
```

In addition, an absolute pathname may be specified with the Multics active function [user name] or [user project]. Such a pathname is not expanded when placed in the search list. It is expanded when first referenced in a user's process. This feature allows search paths to be defined that identify the process directory or home directory of any user.

The "-stop" keyword is used during the initialization of a search list. At all other times it is ignored. It causes the search facility to stop searching for search segments that contain versions of the search list being initialized.

Examples

```
asp dict [hd]>words.dict
```

The absolute pathname ">udd>Project_dir>User_dir>words.dict" is added as a search path to the search list named "dict". This new search path is positioned as the last search path in the "dict" search list.

```
asp compose <insert_files -first
```

The absolute pathname represented by the relative pathname "<insert_files" is added as a search path to the search list named "compose". This new search path is positioned as the first search path in the "compose" search list.

add_search_paths

asp info info_files -after -working_dir

The absolute pathname represented by the relative pathname "info_files" is added as a search path to the search list named "info". This new search path is positioned in the "info" search list after the current search path specified by the keyword "-working_dir".

Name: delete_search_paths, dsp

The delete_search_paths command allows a user to delete one or more search paths from the specified search list.

Usage

delete_search_paths search_list search_paths {control_arg}

where:

1. search_list
   is the name of the search list from which the specified search paths are deleted.

2. search_path1
   specifies a search path to be deleted. The search path may be an absolute or relative pathname or a keyword. In representing the search path it is necessary to use the same name that appears when the print_search_paths command is invoked (without the -exp option).

3 control_arg
   can be the following:

   -all, -a
   specifies that the search list itself is to be deleted. Any search paths specified are ignored.

Name: print_search_paths, psp

The print_search_paths command prints the search paths in the specified search lists.

Usage

print_search_paths {search_lists} {-control_arg}

where:

1. search_list
   is the name of a search list to be printed. If no search list is specified, then all search lists referenced and initialized in this process are printed.

2. control_arg
   can be the following:

   -expanded, -exp
   specifies that all keyword search paths except -stop and -referencing_dir are expanded into their current absolute pathnames. The -stop and -referencing_dir keywords are printed in unexpanded form.

Name: set_search_paths, ssp

The set_search_paths command allows a user to completely define or replace the search paths contained in the specified search list.

Usage

set_search_paths search_list {search_paths}

where:

1. search_list
   is the name of the search list to be set. If this search list does not exist, then it is created.

2. search_path
   is a search path that is placed into the specified search list. The search paths are added in the order in which they are specified in the command line. The search path may be an absolute or relative pathname or a keyword. (For a list of acceptable keywords see the add_search_paths command.) If no search path is entered, then the specified search list is set as if it were being initialized for the first time in the user's process.

Note: If the specified search list already exists, it will be replaced by the specified search paths; if no search paths are given then the search list is replaced with an empty search list. It is an error to create a new empty search list.

where_search_paths

Name: where_search_paths, wsp

The where_search_paths command and active function, given a search list name and a reference name, returns the absolute pathname(s) of where this reference name can be found. The search for the reference name is made using the current search paths contained in the specified search list.

Usage

where_search_paths search_list ref_name {control_arg}

{wsp search_list ref_name {-control_arg}}

where:

1. search_list
   is the name of the search list searched.

2. ref_name
   is the reference name sought.

3. control_arg
   can be the following:

   -all, -a
   specifies that all occurrences of this reference name found by searching this search list should be returned. The default is to return only the first occurence.

search_paths_$find_dir

Name: search_paths_$find_dir

The search_paths_$find_dir entry point, given a search list name and a reference name, returns an absolute pathname of where this reference name can be found. The search for the reference name is made using the current search paths contained in the specified search list.

Usage

```
dcl  search_paths_$find_dir entry (char(*), ptr, char(*),
     char(*), char(*), fixed bin(35));

call search_paths_$find_dir (sl_name, sl_seg_p, ref_name,
     ref_path, dir_name, code);
```

where:

1. sl_name (Input)
   is the name of the search list searched.

2. sl_seg_p (Input)
   is a pointer to the search segment in which the specified search list can be found. If this pointer is null then the process search segment is used.

3. ref_name (Input)
   is the name of the reference name sought.

4. ref_path (Input)
   This pathname is used when processing the search path keyword "-referencing_dir". If this is the pathname of a link, ther the target pathname is used. The directory portion of this pathname (or target pathname) is used as the referencing directory. If ref_path is null or blank, then the "-referencing_dir" search path is skipped.

5. dir_name (Output)
   is the directory portion of the pathname found for the specified reference name.

search_paths_$find_dir

6. code (Output)
is a standard status code. The following subset of possible values are of particular interest:

error_table_$bad_ref_name
illegal reference name, may contain ">" or "<".

error_table_$ref_name_not_found
reference name not found.

Name: search_paths_$find_all

The search_paths_$find_all entry point, given a search_list name and a reference name, returns a copy of the sl_info structure (see search_paths_$get) in which the pathname entries are those pathnames in which the reference name can be found. The search for the reference name is made using the current search paths contained in the specified search list.

Usage:

```
dcl  search_paths_$find_all entry (char(*), ptr, char(*),
     char(*), ptr, fixed bin(35), ptr, fixed bin(35));

call search_paths_$find_all (sl_name, sl_seg_p, ref_name,
     ref_path, sl_info_area_p, sl_version, sl_info_p, code);
```

where:

1. sl_name (Input)
   is the name of the search list searched.

2. sl_seg_p (Input)
   is a pointer to the search segment in which the specified search list can be found. If this pointer is null then the process search segment is used.

3. ref_name (Input)
   is the name of the reference name sought.

4. ref_path (Input)
   is used when processing the search path keyword "-referencing_dir". If this is the pathname of a link, then the target pathname is used. The directory portion of this pathname (or target pathname) is used as the referencing directory. If ref_path is null or blank, then the "-referencing_dir" search path is skipped.

5. sl_info_area_p (Input)
   is a pointer to a user-supplied area into which sl_info can be allocated.

6. sl_version (Input)
   is the version of the sl_info structure desired by the caller. It is used to insure that compatibility with future versions of search can be maintained.

7. si_info_p (Output)
   is a pointer to a structure filled in with those paths in the specified search list in which reference name can be found. The format of this structure can be found in the section on search_paths_$get.

8. code (Output)

Name: search_paths_$get

The search_paths_$get entry point returns the current search paths in the specified search list.

Usage

```
dcl  search_paths_$get entry (char(*), bit(36) unaligned,
     char(*), ptr, ptr, fixed bin(35), ptr, fixed bin(35));

call search_paths_$get (sl_name, sl_control, ref_path, sl_seg_p,
     sl_info_area_p, sl_version, sl_info_p, code);
```

where:

1. sl_name (Input)
   is the name of the search list returned.

2. sl_control (Input)
   is a control mask defined by the structure sl_control_s, given in the notes below. A "1"b bit indicates that the corresponding type of entry should be returned in expanded form.

3. ref_path (Input)
   is the pathname used when processing the search path keyword "-referencing_dir". If this is the pathname of a link, then the target pathname is used. The directory portion of this pathname (or target pathname) is used as the referencing directory. If ref_path is null or blank, then the "-referencing_dir" search path is skipped.

4. sl_seg_p (Input)
   is a pointer to the search segment in which the specified search list can be found. If this pointer is null, then the process search segment is used.

5. sl_info_area_p (Input)
   is a pointer to a user-supplied area into which sl_info can be allocated.

6. sl_version (Input)
   is the version of the sl_info structure desired by the caller. It is used to insure that compatability with future versions of search can be maintained.

7. sl_info_p (Output)
   is a pointer to a structure filled in with the specified search list. The format of this structure is defined in the notes below.

8. code (Output)

Notes

Below is a description of the search list info structure returned by this entry point. This structure is declared in the include file sl_info.incl.pl1.

```
dcl 1 sl_info based aligned based (sl_info_p),
      2 version fixed bin,
      2 num_paths fixed bin,
      2 change_index_p ptr,
      2 change_index fixed bin (71),
      2 pad1 (6) bit (36),
      2 paths (max_num_paths refer (sl_info.num_paths)),
        3 type fixed bin,
        3 pad2 (2) bit (36),
        3 pathname char (168) unal;
```

where:

1. version
   is the number of the version of this structure returned.

2. num_paths
   is the number of search paths contained in this search list.

3. change_index_p
   is a pointer to a status index associated with this search list. The status index is stored as a fixed bin (71) integer. This status index is incremented whenever the search list is modified.

4. change_index
   is the value of the search list status index at the time that sl_info is created. Using the pointer to this index (above), the caller may, at any later time, compare the returned index value with the current index value and thus determine if the search list has been modified

5. type
is an index that specifies the type of the search path. The following values may be returned:

0 => absolute pathname
1 => unexpanded pathname (=> [user xxxx])
2 => the keyword -stop
3 => the keyword -referencing_dir
4 => the keyword -working_dir
5 => the keyword -process_dir
6 => the keyword -home_dir

6. pathname
is the search path.

The argument sl_control is defined by the structure shown below, which is declared in the include file sl_control_s.incl_s.pl1.

```
dcl 1 sl_control_s unaligned based (addr (sl_control)),
      2 af_pathname bit (1),
      2 key_stop bit (1),
      2 key_ref_dir bit (1),
      2 key_work_dir bit (1),
      2 key_proc_dir bit (1),
      2 key_home_dir bit (1),
      2 pad (30) bit (1);
```

Expanding the keyword -stop implies ignoring it. Expanding the keyword -referencing_dir gives back the path name ref_path if ref_path was supplied; otherwise it causes -referencing_path to be ignored. There is no control bit for absolute pathnames as they are the same in expanded and unexpanded form.

Name: search_paths_$list

The entry point search_paths_$list returns a list of the names of all search lists currently defined in the specified search segment.

Usage

```
dcl  search_paths_$list entry (ptr, ptr, fixed bin(35), ptr,
     fixed bin(35));

call search_paths_$list (sl_seg_p, sl_list_area_p,
     sl_list_version, sl_list_p, code);
```

where:

1. sl_seg_p (Input)
   is a pointer to the search segment listed. If this pointer is null, then the process search segment is listed.

2. sl_list_area_p (Input)
   is a pointer to a user-supplied area into which the structure sl_list can be allocated.

3. sl_list_version (Input)
   is the version of the sl_list structure desired by the caller. It is used to insure that compatibility with future versions of search can be maintained.

4. sl_list_p (Output)
   is a pointer to a structure containing a list of search lists. The format of this structure is defined in the notes below.

5. code (Output)

search_paths_$list

Notes

Below is a description of the structure that is used to list the search lists contained in a search segment. This structure is declared in the include file sl_list.incl.pl1.

```
dcl 1 sl_list based (sl_list_p) aligned,
        2 version fixed bin,
        2 num_lists fixed bin,
        2 pad (6) bit (36),
        2 name (max_num_lists refer (sl_list.num_lists))
                char (32) unaligned;
```

where:

1. version
   is the number of the version of this structure returned.

2. num_lists
   is the number of search list names contained in this search segment.

3. names
   is an array of search list names.

search_paths_$set

Name: search_paths_$set

The search_paths_$set entry point sets the specified search list so that it contains the specified search paths.

Usage

```
dcl  search_paths_$set entry (char(*), ptr, ptr, fixed bin(35));

call search_paths_$set (sl_name, sl_seg_p, sl_info_p, code);
```

where:

1. sl_name (Input)
   is the name of the search list set.

2. sl_seg_p (Input)
   is a pointer to the search segment in which the specified search list can be found. If this pointer is null, then the process search segment is used.

3. sl_info_p (Input)
   is a pointer to a caller supplied structure that contains the search paths being set into the specified search list. (See the search_paths_$get entry point for a description of this structure.)

4. code (Output)

Note: If no search paths are specified and the specified search list exists, then all search paths are deleted from the search list. It is an error to create a new empty search list.

search_paths_$delete_list

Name: search_paths_$delete_list

The search_paths_$delete_list entry point deletes a specified search list from the specified search segment.

Usage

```
dcl  search_paths_$delete_list entry (char(*), ptr,
     fixed bin(35));

call search_paths_$delete_list (sl_name, seg_p, code);
```

where:

1. sl_name (Input)
   is the name of the search list to be deleted.

2. seg_p (Input)
   is a pointer to the search segment in which the specified list can be found. If this pointer is null, the process search segment will be used.

3. code (Output)

search_paths_$delete_path

Name: search_paths_$delete_path

The search_paths_$delete_path entry point deletes a specified set of search paths from the specified search list.

Usage

```
dcl  search_paths_$delete_path (char(*), ptr, ptr,
     fixed bin(35));

call search_paths_$delete_path (sl_name, seg_p, sl_info_p, code);
```

where:

1. sl_name (Input)
   is the name of the search list from which the paths are to be deleted.

2. seg_p (Input)
   is a pointer to the search segment in which the specified list can be found. If this pointer is null, the process search segment will be used.

3. sl_info_p (Input)
   is a pointer to a user supplied structure containing the names of the paths to be deleted (see "get"). The user must fill in the version, num_paths, and paths.pathname entries. On return this structure will contain only the pathnames which were passed for deletion but were not found in the initiated search list. Num_paths will contain the number of paths which could not be found but for which deletion was requested.

4. code (Output)