①PC- Roach

To: Distribution

From: Bill Silver

Date: April 13, 1977

Subject: Command Search Facility

INTRODUCTION

This memorandum describes a proposed new command search facility. It is a revival of ideas previously proposed by Txom McGary in MTB-112.

This new 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: Bill Silver
Honeywell Information Systems
575 Tech. Sq.
Cambridge, Mass. 02139

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

Silver.Multics

or call: (617) 492-9300
HVN 261-9306

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 her own list of directories being searched, gives the Multics user an unparalleled ability to dynamically customize her 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. A user must separately initialize each search facility that she will use. In addition, 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 be 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 xxxx]. 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 process directory of that user.

   -continue
   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.

   -initiated_segments
   is a keyword that does not represent a pathname. Segments will be found via this search path if they are currently initiated.

   -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. Search segments are identified by entrynames with a last component that is the suffix "search". All search lists residing in the same search segment must have different names. Search lists with the same name may reside in different search segments. All of the commands provided to manage the search lists of a process can, by using a special control argument, be used to manage a search list in any search segment.

4. Process Search Segment: contains the current, temporary representations of all search lists that have been referenced by the process. The use of process search segments will be transparent to the user. These search segments will reside in the process directory. All of the commands provided to manage a search list will, by default, operate on the search lists contained in the process search segment.

Process Search List Initialization

The new search facility will use its own searching capabilities to initialize a search list the first time it is referenced in a process. Initializing a search list for a process involves adding that search list to the process search segment. A search list will be added to the process search segment by copying it from other search segments.

The new search facility will use its own search list, named "search". The "search" search list contains a list of search paths that are pathnames of search segments. The search segments specified in the "search" search list will be tested to determine if they contain a version of the search list being initiated. The search segments will be tested in the order that they appear in the "search" search list. The search list being initiated will be copied from the first search segment found to have a version of that search list.

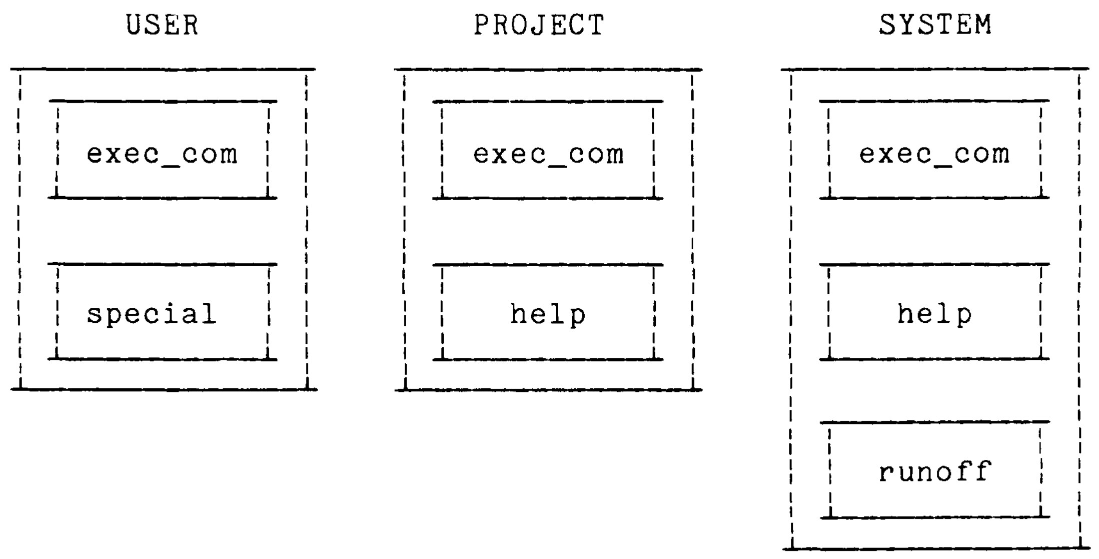
If the search lists being copied contains the search path keyword "-continue", then the "search" for versions of this search list will continue. If another search segment is found that contains a version of the search list being initialized, then any search paths contained in this new version, that have not already been copied into the search list being initialized, will be copied. This procedure will be repeated until no more versions of this search list can be found, or until a search list is found that does not contain the "-continue" keyword.

The "search" search list must itself be initialized in a process. In fact, it will always be the first search list initialized and copied into the process search segment. This is done by using the following simple "meta" search list:

> Meta Search List: The "search" search list will be initialized in every process by copying it from the system search segment: >sc1>system.search

The system search segment ">sc1>system.search" will also contain search lists that define the site defaults for all standard search lists. Project administrators may install a project search segment that contains search lists that are different, or in addition to, the search lists contained in the system search segment. Individual users may do the same. A user may have his search lists initialized from any hierarchy of search segments simply by adding the desired search segments to his "search" search list.

The following example shows how search segments can be used to initialize search lists in a process. Assume that there exist user, process, and system search segments, and that they contain the search lists as shown below:



Now assume that the user's "search" search list specifies these search segments, i.e., contains the following search paths:

```
>udd>Project_id>User_id>user.search
>udd>Project_id>project.search
>sc1>system.search
```

The user will thus have her search lists initiated from the following search segments:

SEARCH LIST	SEARCH SEGMENT
exec_com	user.search
help	project.search
runoff	system.search
special	user.search

Benefits of the New 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.

4. *Efficient Storage*: Although many standard search lists will be provided, each process will have to initialize and copy into its process search segment only those search lists that it actually uses. Projects and users that maintain their own private search segments, will need to keep in their search segments only their private search lists and those standard search lists for which they want a different set of search paths.

5. Automatic Tracking of System Defaults: A project or user that maintains their own search segments may, if they use the "-continue" search path keyword, automatically pick up any changes made to the site defaults of a search list. Thus when a system administrator adds a new search path to a standard search list, that search path may also be added to private versions of that search list.

6. 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. It is easy to learn to maintain search segments because this is done by using the same commands that are used to maintain a process' current search lists.

7. Easy to Use: No longer will a user have to issue several commands to initialize the command searching to be performed in her process. One command to set the "search" search list will effectively initialize all search lists that will be used by the process.

8. 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.

9. 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 then 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:

<u>Language</u> <u>Processors</u>: 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.

<u>exec_com</u>: 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.

<u>Text</u> <u>Editors</u>: The Multics text editors that provide macro capabilities (qedx, ted, teco) could use the new search facility to make it convenient to use editor macro libraries.

<u>Help</u>: 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.

<u>Runoff</u>: The "runoff" command will be able to have its own search list. It would no longer have to use a search facility designed for the Multics language processors.

<u>Multics</u> <u>System</u> <u>Tapes</u>: 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.

<u>Home</u> <u>Directory</u> <u>Dependants</u>: The "debug" and "mail" commands could use the new search facility to specify the default location for "break segments" and "mailboxes".

<u>Abbrev</u>: 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.

<u>MPM</u> <u>Documentation</u>

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

Name: add_search_paths, asp

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

Usage

```
add_search_paths search_list search_paths {-control_args}
```

where:

1. search_list
   is the name of the search list to which the new search paths will be added. If this search list is not already initialized for this process, then it will be initialized and the command will proceed normally. If this search list cannot be initialized, then the command will be terminated with an error.

2. search_path*i*
   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 will be 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 not necessary to use the same name that appears when the print_search_paths command is invoked. Any equivalent representation of a current search path is acceptable.

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

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

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

3. control_arg
can be the following:

-segment path, -sm path
specifies that the search list to be updated is contained in the search segment specified by path. If the last component of path is not ".search", then that suffix will be assumed. If the specified search segment does not exist, or if the specified search list is not contained in this search segment, then the command will be terminated with an error.

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.

-continue
-home_dir
-initiated_segments
-process_dir
-referencing_dir
-working_dir

In addition, an absolute pathname may be specified with the Multics active function [user xxxx]. Such a pathname will not be expanded when it placed in the search list. It will be 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 "-continue" keyword is used during the initialization of a search list. At all other times it is ignored. It causes the search facility to continue searching for search segments that contain versions of the search list being initialized.

The "-initiated_segments" keyword causes the search facility to find a segment if it is currently initiated.

Examples

```
asp info [hd]>info
```

The absolute pathname ">udd>Project_dir>User_dir>info" will be added as a search path to the search list named "info". This new search path will be positioned as the last search path in the "info" search list.

```
asp runoff <insert_files -first
```

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

```
asp exec_com library -after -working_dir
```

The absolute pathname represented by the relative pathname "library" will be added as a search path to the search list named "exec_com". This new search path will be positioned in the "exec_com" search list after the current search path specified by the keyword "-working_dir".

```
asp pl1 ">udd>[user project]>incl" -be >ldd>incl -sm >sc1>sys.search
```

The unexpanded pathname ">udd>[user project]>incl" will be added to the search list named "pl1" in the search segment ">sc1>sys.search". This new search path will be positioned before the current search path ">ldd>include". When the "pl1" search list is initialized in a user's process, this search path will be expanded to reference the user's project "incl" directory.

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_args}
```

where:

1. search_list
   is the name of the search list from which the specified search paths will be deleted. If this search list is not already initialized for this process, then it will be initialized and the command will proceed normally. If this search list cannot be initialized, then the command will be terminated with an error.

2. search_path*i*
   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 not necessary to use the same name that appears when the print_search_paths command is invoked. Any equivalent representation is acceptable.

3. control_arg
   can be chosen from the following:

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

   -segment path, -sm path
   specifies that the search list to be updated is contained in the search segment specified by path. If the last component of path is not ".search", then that suffix will be assumed. If the specified search segment does not exists, or if the specified search list is not contained in this search segment, then the command will be terminated with an error.

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_args}
```

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 will be printed. If a search list is specified that is not already initialized for this process, then it will be initialized and the command will proceed normally. If this search list cannot be initialized, then the user will be informed that this search list cannot be found.

2. control_arg
   can be chosen from the following:

   -expanded, -exp
   specifies that all keyword search paths are to be expanded into their current absolute pathnames.

   -segment path, -sm path
   specifies that the search lists to be printed are contained in the search segment specified by path. If no search lists were specified, then all search lists contained in this search segment will be printed. If the last component of path is not ".search", then that suffix will be assumed. If the specified search segment does not exist, then the command will be terminated with an error. If a specified search list is not contained in this search segment, then the user will be informed that this search list cannot be found.

Name: set_search_paths, ssp

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

Usage

```
set_search_paths search_list {search_paths} {-control_arg}
```

where:

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

2. search_path*i*
   is a search path that will be added to the specified search list. The search paths will be 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 will be set as if it were being initialized for the first time in the user's process.

3. control_arg
   can be the following:

   -segment path, -sm path
   specifies that the search list to be set is contained in the search segment specified by path. If the last component of path is not ".search", then that suffix will be assumed. If this search segment does not exist, then the user will be asked if it should be created.

Name: where_search_paths, wsp

The where_search_paths command and active function, given a search list name and a reference name, will return the absolute pathname(s) of where this reference name can be found. The search for the reference name will be 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]
```

where:

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

2. ref_name
   is the reference name to be found.

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.

Name: search_paths_$find

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

Usage

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

call search_paths_$find (sl_name, ref_name, ref_path, dir_name,
     entryname, code);
```

where:

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

2. ref_name (Input)
   is the name of the reference name to be found.

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

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

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

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_$get

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

Usage

```
dcl  search_paths_$get entry (char(*), ptr, ptr, fixed bin(19),
     ptr, fixed bin(71), fixed bin(35));

call search_paths_$get (sl_name, sl_seg_ptr, sl_info_ptr,
     sl_info_size, sl_index_ptr, sl_index, code);
```

where:

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

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

3. sl_info_ptr (Input)
   is a pointer to a caller supplied buffer in which a search list info structure will be returned. The format of this structure is defined in the notes below.

4. sl_info_size (Input)
   is the size (in words) of the caller's search list info buffer. If the size of this buffer is not sufficient to accomodate all of the information that may be returned, then an error will occur.

5. sl_index_ptr (Output)
   is a pointer to a status index associated with this search list. This status index is incremented whenever the search list is modified.

6. sl_index (Output)
   is the value of the search list status index at the time the information about this search list is returned. Using the pointer to this index, the caller may, at any later time, easily compare the returned index value with the current index value and thus determine if the search list has been modified.

7. code (Output)

Notes

Below is a description of the search list info structure returned by this entry point:

```
dcl 1 sl_info  based(sl_info_ptr)  aligned,
      2  version    fixed bin,      /* 1. */
      2  num_spaths fixed bin,      /* 2. */
      2  pad(6)     bit(36),
      2  spaths  (cur_num_spaths refer(sl_info.num_spaths))
                    like sp_info;

dcl 1 sp_info  based(sp_info_ptr)  aligned,
      2  sp_type    fixed bin,      /* 3. */
      2  sp_len     fixed bin,      /* 4. */
      2  pad(2)     bit(36),
      2  spath      char(168);      /* 5. */
```

where:

1. version
   is the current version number of this structure. The caller must set this field to 1.

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

3. sp_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 -continue
3  =>  the keyword -initiated_segments
4  =>  the keyword -referencing_dir
5  =>  the keyword -working_dir
6  =>  the keyword -process_dir
7  =>  the keyword -home_dir
```

Name: search_paths_$get_expanded

The search_paths_$get_expanded entry point will return the current search paths in the specified search list. Any search paths that are unexpanded pathnames or keywords will be expanded into absolute pathnames.

Usage

```
dcl  search_paths_$get_expanded entry (char(*), ptr, ptr,
     fixed bin(19), fixed bin (35));

call search_paths_$get_expanded (sl_name, sl_seg_ptr,
     sl_info_ptr, sl_info_size, code);
```

where:

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

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

3. sl_info_ptr (Input)
   is a pointer to a caller supplied buffer in which a search list info structure will be returned. (See the search_paths_$get entry point for a description of this structure.)

4. sl_info_size (Input)
   is the size (in words) of the caller's search list info buffer. If the size of this buffer is not sufficient to accomodate all of the information that may be returned, then a error will occur.

5. code (Output)

Notes

The special keywords "-continue", "-initiated_segments", and "-referencing-dir" will not be expanded. They will be returned in their keyword format.

Name: search_paths_$list

The entry point search_paths_$list will return a list of the names of all search lists current defined in the specified search segment.

Usage

```
dcl  search_paths_$list entry (ptr, ptr, fixed bin(19),
     fixed bin(35));

call search_paths_$list (sl_seg_ptr, sl_list_ptr, sl_list_size,
     code);
```

where:

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

2. sl_list_ptr (Input)
   is a pointer to a caller supplied buffer in which a structure containing a list of search lists will be returned. The format of this structure is defined in the notes below.

3. sl_list_size (Input)
   is the size (in words) of the caller's search list list buffer. If the size of this buffer is not sufficient to accomodate all of the information that may be returned, then an error will occur.

4. code (Output)

Notes

Below is a description of the structure that is used to list the search lists contained in a search segment:

```
dcl 1 sl_list  based(sl_list_ptr)  aligned,
      2 version    fixed bin,      /* 1. */
      2 num_slists fixed bin,      /* 2. */
      2 pad(6)     bit(36),
      2 slists  (cur_num_slists refer(sl_list.num_slists))
                   char(32);       /* 3. */
```

where:

1. version
   is the current version number of this structure. The caller must set this field to 1.

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

3. slists
   is an array of search list names.

Name: search_paths_$set

The search_paths_$set entry point will set 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_ptr, sl_info_ptr, code);
```

where:

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

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

3. sl_info_ptr (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)