TO: MTB Distribution

FROM: Txom McGary

SUBJECT: Consolidated Search Facility

DATE: August 1, 1974

This document reviews existing search mechanisms used by Standard Service System commands, and proposes a single extensible facility to replace them. There are currently several commands which use search rules of one kind or another, but their treatment of search rules is inconsistent. Include (translator) search rules are maintained by one subroutine, but info file search rules are maintained in two commands (help and check_info_segs). A consolidated facility would be easier to learn, easier to use and could provide functions not currently availible.

Please address comments or questions to Txom McGary, by Multics mail to McGary.RDMS, by Interdepartmental Mail to 39-411, or by telephone at 253-4107.

Existing Search Rule / Search Procedure Mechanisms

1) Object Segment Search Rules:

a) invocation: calls to hcs_$make_ptr or linkage fault.
b) manipulation: by commands set_search_rules, print_search_rules, set_search_dirs, add_search_rules, delete_search_rules; by subroutines hcs_$initiate_search_rules, hcs_$get_search_rules, and parse_search_rules_.
c) default rules: initiated_segments, referencing_dir, working_dir, system_libraries.
d) data base: an array of pointers to directory segments is maintained by the hardcore in the Known Segment Table (KST). (Philippe Janson proposes that this be changed to an array of search rules in the header of the Combined Linkage Segment in MTB-035 "Design of a Dynamic Linker Running Outside the Multics Security Kernal"; refer also to RFC-49, "A Proposal for Removing Name Space Management from Ring Zero", by Richard G. Bratt.)
e) keywords: initiated_segments, referencing_dir, working_dir, home_dir, process_dir, system_libraries, default (must be only search rule given) and set_search_directory (following rules are to be appended after woring_dir rule, used only by set_search_dirs).
f) maximum number of search rules: 21.

2) Include (Translator) Search Rules:

a) invocation: by call to find_include_file_$initiate_count resulting from pl1, alm, or fortran statement "%include x", runoff control line ".if y", or bcpl statement "get z".
b) manipulation: calls to find_include_file_$get_search_rules or find_include_file_$set_search_rules. (Steve Webber has commands ptsr (for print translator search rules) and stsr (set translator search rules) which perform the "get" and "set" functions from command level.)
c) default: working_dir, >udd> user project >include (if it exists) and >ldd>include.
d) data base: internal static structure of find_include_file_ for default search rules. If a user changes from the default, a structure is allocated in the system free storage area.
e) keywords: working_dir, referencing_dir, home_dir, default. ("default" re-establishes the default mentioned in c.)
f) maximum number of search rules: unrestricted.

3) Info File Search Rules:

a) invocation: commands help and check_info_segs. (Also, a new command called list_help was recently added to the AML.)
b) manipulation: none.

c) default: >doc>iml_info_segs and >doc>info.
d) data bases: array of char(168) strings in help and cis.
e) keywords: does not apply.
f) maximum number of search rules: does not apply.

4) Peruse Text Search:

a) invocation: peruse_text command.
b) manipulation: none.
c) default: >doc>pt.
d) data base: character string in persue_text program.
e) keywords: does not apply.
f) maximum number of search rules: does not apply.

5) Macro Search Rules:

a) invocation: use of macro-interpreters exec_com, qedx and teco.
b) manipulation: none for system commands, RDMS version of exec_com has own search rules, SIPB version uses object search rules. The Author Maintained macro-interpreter and editor teco has its own search rules: working_dir, home_dir and TECO-dir (>system_library_auth_maint), and the user may replace home_dir with an arbitrary directory pathname.
c) default: does not apply for system. (RDMS exec_com default includes RDMS project's ec dir, >udd>RDMS>service.ec. SIPB default is default for object segs. Teco default is working_dir, home_dir, and TECO-dir.)
d) data_base: does not apply for system, RDMS exec_com has internal static structure with room for up to 16 rules. SIPB exec_com uses object segment search rules, so data base is in the KST. Teco uses an internal static array of three pathnames.
e) keywords: does not apply for system, RDMS subroutine recognizes keywords for object segments, SIPB version uses object search rules.
f) maximum number of search rules: does not apply for system exec_com, RDMS exec_com allows up to 16 rules, SIPB version allows up to 21 (using object search rules).

6) Other Search Facilities:

I am not aware of any other search mechanisms in the Standard Service System, but elsewhere search procedures are invoked.

-- get_library_segment is a system tool for getting a copy of a segment stored in the Multics online libraries as a component of an archive. This program allows a search procedure to be associated with each directory, as specified in an ascii text control file.
-- generate_mst is a tool for generating Multics System Tapes

(MST's). Given an ascii text segment specifying segments to be included on the system tape, the program searches the working directory and >ldd>hard>object for the segments named. Alternately, a set of search rules defined by an ascii text segment containing directory pathnames may be used.

-- library_descriptor and library_map, tools Gary Dixon is developing for management of the Multics libraries. A search procedure is associated with each directory or subtree of directories. The programs have been designed so they can be used on library hierarchies besides the system's.

-- debug uses a search scheme to locate the source segment of an object segment when the working segment is an object segment and the source segment is referenced ("&a123,s5").

These search mechanisms do not fit well into the proposed facility, but omitting them seems reasonable since they are of a sufficiently distinct character that they should be managed separately. (For example, get_library_segment's search is tied to the structure of the directories searched rather than simply being a search for a segment name.)

Deficiencies in the Current Search Facilities

A) Subsystems cannot use the system "help" and "cis" commands conveniently if they have their own info library. Many subsystems maintain their own version of help and cis, probably for this reason. (For RDMS at least, that is the reason.)

B) The current facility for exec_com, qedx, etc. makes it impossible for a project, subsystem or individual user to establish a macro library which can be used conveniently.

C) The current facilities are not extensible (with the exception of translator search rules which could accomodate new translators).

D) The current facilities perform the same function in an inconsistent fashion. A consolidated treatment would be simpler to learn and more efficient to use, and would eliminate some duplicated code. Additionally, some subsystem programs would no longer be needed if their function were performed by the standard system.

E) A consolidated facility could be documented in one place, instead of including a description of search facilities in the write up of each translator, macro-interpreter, or info command. (The write up for each would describe the default applicable and refer to the documentation of the search facility.)

Overview of Proposed New Search Facility

I propose that the existing search mechanisms mentioned in paragraphs 1 through 5 above be consolidated into a single search facility subroutine (to be named "search_facility_"). All data bases, defaults and keywords related to search rules would be maintained by this procedure. The procedure would allow new search groups to be established dynamically. Existing search mechanisms would be supported by the new facility in a fashion transparent to the user. (Object search rules would of course still be maintained as an array of directory pointers in the KST, but getting or setting object search rules would be accomplished via search_facility_. Include search rules would function exactly as they do presently, but find_include_file_ would be a writearound to search_facility_. For the sake of efficiency it could be an entry in the same procedure with search_facility_. Check_info_segs would call the facility to get a list of directory pathnames instead of maintaining a list of directories internally. Help would call the facility to find info files. Macro-interpreting programs would be modified to call search_facility_ instead of expand_path_ and hcs_$initiate_count. Peruse_text would be modified to call the facility. If the facility were invoked to find some segment, and the user had not explicitly established his or her own search rules for the search suffix in question, the defalts would be as follows: object segments would continue to use the current defaults, include files would use current defaults, help and cis would get as their default the directoreies >doc>iml_info_segs and >doc>info, for peruse_test the default would be >doc>pt, and teco would get working_dir, home_dir and TECO-dir. For all other groups the default would be "working_dir", so for the macro-interpreters other than teco this would be equivalent to the current situation.)

Detailed Proposal of New Subroutine

The subroutine search_facility_ would have the entry points described below.

Entry: search_facility_$set

Given an array of char(168) strings containing absolute directory pathnames and keywords, this entry establishes those search rules for a suffix or suffixes.

```
dcl search_facility_$set ext entry (ptr, fixed bin, ptr, fixed
     bin, fixed bin(35));
dcl suffix_array(N) char(32) aligned;
dcl search_rule_array(M) char(168) aligned;
call search_facility_$set (addr(suffix_array), num_suffixes,
```

```
     addr(search_rule_array), num_rules, code);
```

This is the only place that grouping of suffixes is specified.

Entry: search_facility_$get

This entry point would fill in a user supplied rules array with absolute directory pathnames and keywords.

```
dcl search_facility_$get ext entry (char(32), ptr, fixed bin,
     fixed bin(35));
dcl search_rule_array(20) char(168) aligned;
call search_facility_$get (suffix, addr(user_rule_array),
     num_rules, code);
```

Upon successful return from the subroutine, code would be 0, num_rules would be set to the number of rules currently active for the search suffix, and the first num_rules rules of user_rule_array would contain the rules (keywords would appear where they were used in setting rules). Otherwise, if there were more rules active than could fit in user_rules, code would be set to error_table_$too_many_sr, num_rules would be set to the number of rules active so the calling program could allocate more space and try again, and user_rule_array would not be modified. If no search rules were active for the suffix supplied, a non zero code would be returned but user_rule_array would contain the default (working_dir) and num_rules would be set to the number of default rules. A new error_table_ code for "search suffix not found" would be needed.

Entry: search_facility_$star_get

This entry could be used to get all currently active search suffixes and associated rules, or only the rules for a subset of the suffixes with active rules, the subset being specified by star names.

```
dcl search_facility_$star_get ext entry (ptr, fixed bin, ptr,
     fixed bin, fixed bin, fixed bin, ptr, ptr, ptr, fixed
     bin(35));
call search_facility_$star_get (addr(star_name_array),
     num_star_names, area_ptr, num_groups, num_suffixes,
     num_rules, group_ptr, suffix_ptr, rule_ptr, code);
dcl 1 group(num_groups) based (group_ptr),
      2 rule_index fixed bin,
      2 rule_count fixed bin,
      2 suffix_index fixed bin,
      2 suffix_count fixed bin;
dcl suffix(num_suffixes) char(32) based (suffix_ptr);
dcl rule(num_rules) char(168) based(rule_ptr);
```

The calling program would supply an area and pass a pointer to it. The current search suffixes would be checked against the star names in star_name_array (using match_star_name_), and qualifying suffixes would be output in suffix_array which would be allocated in the supplied area. Each structure "group(i)" would describe a set of rules for the group, starting at rule (group(i).rule_index) and continuing for group(i).rule_count rules. The suffixes starting at suffix_array (group(i).suffix_index) going for group(i).suffix_count suffixes would be using those rules. This would make it easy to print rules:

```
ec:
qedx:
          working_dir,
          >udd>RDMS>macros;

incl.pl1:
incl.fortran:
incl.alm:
bcpl:
runoff:
          working_dir,
          >udd>RDMS>include,
          >ldd>include;
```

and so on.

The array of group structure returned for the situation above would be:

```
group(1):
          rule_index = 1;
          rule_count = 2;
          suffix_index = 1;
          suffix_count = 2;
group(2):
          rule_index = 3;
          rule_count = 3;
          suffix_index = 3;
          suffix_count = 5;
```

Entry: search_facility_$expanded_get

This entry would fill in a search rules array like that described for search_facility_$get_search_rules, but no keywords are returned. Any keywords in the current search rules for the search group are expanded into the directory pathname(s) implied. This entry is intended for use by commands such as check_info_segs which do something for each directory of the current search rules, instead of finding one segment using those

rules.

```
dcl search_facility_$expanded_get ext entry (char(32), ptr, fixed
     bin, fixed bin(35));
call search_facility_$expanded_get (suffix, addr(user_rules),
     num_rules, code);
```

Entry: search_facility_$initiate_count

Given a suffix and a search_name, this entry would attempt to find a segment named search_name in the directories of the search rules for suffix. A "referencing_ptr" would be passed for use if the referencing_dir rule were encountered. When searching at the top level, this pointer would be null (as for finding a command or a macro named in the command line invoking a macro-interpreter). The pointer would be non-null when a second level search would be performed: a command calls a subroutine, so referencing_ptr points to the segment containing the command; or an include file includes a lower level file, or a macro invokes a macro.

```
dcl search_facility_$initiate_count ext entry (char(32), ptr,
     char(32), ptr, fixed bin(24),  fixed bin(35));
call search_facility_$initiate_count (suffix, referencing_ptr,
     search_name, seg_ptr, bit_count, code);
```

Keywords

Search rules are either absolute directory pathnmes or keywords. Absolute pathnames always begin with the greater-than character (">"). Keywords can be divided into two groups: those computed once per process, and those computed once for each use.

Group One: Compute Once per Process

```
home_dir
process_dir
system_libraries (object search rules only)
```

Group Two: Compute at Each Use

```
working_dir
referencing_dir
initiated_segments (object search rules only)
```

The rules are self explanatory. For upwards compatability, the rule "system_libraries" must be recognized for object segment search. The rule "initiated_segments" must be recognized for object segment search also, even though it is not really a rule. (It is a reminder that the hardcore requires that the KST be

searched before any other rule is invoked, but this "rule" must always appear first and cannot be turned off.)

New Commands

The functions of existing search rule manipulating commands "set_search_rules", "print_search_rules", "add_search_rules", "delete_search_rules" and "add_search_dirs" will be taken over by a single new command "searcher" (short name "sh"). The old commands will become entries in this new command.

Entry: searcher
sh

Usage: searcher <subcommand>
or
sh <subcommand>

where <subcommand> is any of the following:

1) To replace current rules for a suffix:

searcher replace <suffix> <rule1> . . . <ruleN>

2) To add a rule or rules to the current rules for a given suffix:

searcher add <suffix> <rule1> . . . <ruleN>

Rules <rule1> through <ruleN> are added at end of the current rules for <suffix>.

searcher add <suffix> <rule1> ... <ruleN> -after <existing_rule>

"-after" is recognised by the add subcommand as having a special meanrng. <existing_rule> must be in the search rules for <suffix>. In the new rules for <suffix>, rules <rule1> through <ruleN> are appended after <existing_rule>.

searcher add <suffix> <rule1> ... <ruleN> -before <existing_rule>

"-before" is also recognized as having a special meaning. In the new rules, <rule1> . . . <ruleN> are inserted before <existing_rule>.

searcher add <rule1> <rule2> -after <existing_rule1> <rule3> <rule4> <rule5> -before <existing_rule2> <rule6> <rule7>

The three types of add subcommand can be combined. The above command appends <rule1> and <rule2> after <existing_rule1>, inserts <rule3>, <rule4> and <rule5> before <existing_rule2>, and appends <rule6> and <rule7> after the end of the rules.

3) To delete a rule or rules from the search rules for a suffix:

searcher delete <suffix> <rule1> . . . <ruleN>

4) To print the search rules active for a suffix, or to print all current rules:

searcher print <suffix1> . . . <suffixN>

The rules for the suffix_i mentioned will be printed. If no suffix_i are specified, all rules currently active are printed with associated suffixes (see search_facility_$star_get above for an example).

5) To initialize rules for all suffixes:

searcher init <path>

<path> is the pathname of an ascii text segment. The entryname part of path must end in ".sh. If ".sh" is not the last component of the entryname portion of path, it is added. If no path is specified, it is assumed to be >udd> user project > user name > user name .sh (for example, >udd>RDMS>McGary>McGary.sh).

The ascii text segment contains rule specifications in the form:

```
 <suffix>:
. . .
<suffix>:  <rule1>,
           . . . ,
           <ruleN>;
```

Whitespace is ignored. For example, to initialize tranlator (include) search rules and macro-interpreter search rules, a user might create an ascii text file named user name .sh in his home directory, and this text file would contain:

```
incl.pl1:
incl.alm:
incl.fortran:
bcpl:
runoff:
          working_dir,
          >udd>RDMS>McGary>version2.include,
          >udd>RDMS>include,
          >ldd>include;
ec:
qedx:
teco:
          working_dir,
```

```
>udd>RDMS>macros,
>am;
```

Note that the input form required is the same as that which would be produced by "searcher print", which dumps all search rules.

New Active Function

Finally, a command / active-function which searches for a segment using the search rules of a search suffix would be useful.

Entry: searcher_where
swh

Usage:
```
searcher_where <suffix> name1 . . . nameN
     or
 searcher_where <suffix> name1 . . . nameN
     or
swh <suffix> name1 . . . nameN
     or
 swh <suffix> name1 . . . nameN
```

For each name*i*, search_facility_$initiate_count is called. If the invocation was as a command, and the name is found, the full pathname, including the primary name on the segment, is printed. If a name*i* is not found an error message is printed. If the invocation was as an active function, the return value is formed by concatenating together the pathnames found for each name*i* separated by blanks. If a name*i* was not found during active function invocation, active_fnc_error_ is invoked unless the control argument "-no_error" or "-ne" appeared in the argument list. In that case no pathname is appended to the end of the return string for the name*i*, and the program continues processing with the next name*i*. (This allows an exec_com or absin statement such as

```
&if equal swh info x -noerr "" &then . . .
```

which checks if a segment "x.info" is encountered is in the "info" search path.) If a name*i* does not end in <suffix>, <suffix> is appended for the search. Either the command or active function forms also take the control arguments "-directory" or "-d" (return only the directory portion of the full pathanme) and "-terminate" or "-t" (terminate a single null reference name from the segment if it was initiated).

Overview of Internal Operation of search facility

Design Principles:

1) Installation of the new facility should not degrade the performance of the old interface that become writearounds.
2) There should be no extra overhead for a user who does not use the new features.
3) The new facility should be reasonably efficient.
4) At no time should a user get an inconsistent set of search rules (half old, half new if interrupted while resetting).

A Possible Implementation:

Each distinct set of search rules would be recorded in a "Rules" structure:

```
dcl 1 Rules aligned based,
      2 allocated bit(1) init("1"b),
      2 usage_count fixed bin,
      2 num_active fixed bin,
      2 rule(20),
        3 opcode fixed bin,
        3 path char(168);
```

If opcode were 0, path would be a directory pathname. Otherwise, opcode indicates a keyword rule: 1 -> working_dir, 2 -> referencing_dir, 3 -> home_dir, 4 -> process_dir. When the initiate_count entry is invoked, it uses opcode to index in a transfer vector:

```
          go to op(opcode);

     op(0):
          dir = path;

     make_call:
          call hcs_$initiate_count(dir, search_name, "",
               bit_count, 0, seg_ptr, code);
          if seg_ptr ¬= null() then return;
          else go to next_rule;

     op(1):
          call get_wdir_(dir);
          go to make_call;

     op(2):
          . . .
          etc.
```

The bit switch "allocated" would be used to remember if the structure is internal static, or had been allocated in the system free area and might be freed. "usage_count" would record the

number of suffixes associated with the Rules. When usage_count is reduced to zero, the structure may be freed.

There would be an internal static version of Rules recording the defaults for the currently recognised suffixes. The suffixes "incl.pl1", "incl.alm", "incl.fortran", "runoff" and "bcpl" would have the default rules of Static_Include_Rules, "pt" would have Static_pt_Rules, "info" would have Static_info_Rules, and "teco" would have Static_teco_Rules. Since the current default for the system version of exec_com and qedx is the working_dir, both could use Static_wd_Rule.

The rules for a search suffix would be found by looking through a "Table" structure:

```
dcl 1 Table aligned based,
      2 allocated bit(1) init("1"b),
      2 next_Table_ptr ptr,
      2 num_active fixed bin,
      2 binding(20),
        3 suffix char(32),
        3 rule_ptr ptr;
```

An internal static version of this structure would bind the currently recognised suffixes to the appropriate default structures:

```
dcl 1 Static_Table aligned internal static,
      2 allocated bit(1) initial("0"b),
      2 next_Table_ptr ptr initial(null()),
      2 binding(20),
        3 suffix char(32) initial("incl.pl1", "incl.alm",
                                  "incl.fortran",
                                  "runoff", "bcpl",
                                  "info",
                                  "pt",
                                  "teco",
                                  "qedx",
                                  "ec",
                                  (10) (1) ""),
        3 rule_ptr ptr initial ( (5) addr(Static_Include_Rules),
                                 addr(Static_info_Rules),
                                 addr(Static_pt_Rules),
                                 addr(Static_teco_rules),
                                 (2) addr(Static_wd_rule),
                                 (10) null());
```

If a user changed from the default rules by setting search rules for any suffix, a new Table would be allocted and appropriate Rules structures would be allocated and filled in. At the last moment, when the new Table and Rules structures were consistent with each other, an internal static pointer "first_table_ptr" (which would be initial addr(Static_Table),)

would be set to the address of the new Table. Then the old Table and Rules could be freed. In this way a user would never receive an inconsistent set of rules.

If a user sets search rules for more than 20 suffixes, a second (or third) Table would be allocated, and the next_Table_ptr of the first Table would be set to the address of the second Table.

(END)